DOTTORATO DI RICERCA IN INFORMATICA
IX CICLO NUOVA SERIE
UNIVERSITA' DEGLI STUDI DI SALERNO

On the Verification of Parametric and Real-Time Systems

Barbara Di Giampaolo

November, 2010

PhD Program Chair
Prof.
Margherita Napoli

Supervisor
Prof.
Margherita Napoli

1. PhD Program Chair: Prof. Margherita Napoli

2. PhD Committee: Prof. Alfredo De Santis, Prof. Marco Faella, Prof. Domenico Talia.

3. Supervisor: Prof. Margherita Napoli

Day of the defense: April 29th, 2011

Signature from head of PhD committee:

Abstract

Parametric and Real-Time Systems play a central role in the theory underlying the *Verification* and *Synthesis* problems.

Real-time systems are present everywhere and are used in safety critical applications, such as flight controllers. Failures in such systems can be very expensive and even life threatening and, moreover, they are quite hard to design and verify. For these reasons, the development of formal methods for the modeling and analysis of safety-critical systems is an active area of computer science research.

The standard formalism used to specify the wished behaviour of a real-time system is *temporal logic*. Traditional temporal logics, such as linear temporal logic (LTL), allow only qualitative assertions about the temporal ordering of events. However, in several circumstances, for assessing the efficiency of the system being modeled, it may be useful to have additional quantitative guarantees. An extension of LTL with a real-time semantics is given by the *Metric Interval Temporal Logic* (MITL), where changes of truth values happen according to a splitting of the line of non-negative reals into intervals.

However, even with quantitative temporal logics, we would actually like to find out what quantitative bounds can be placed on the logic operators.

In this thesis we face with the above problem proposing a parametric extension of MITL, that is the *parametric metric interval temporal logic* (PMITL), which allows to introduce parameters within intervals . For this logic, we study decision problems which are the analogous of satisfiability, validity and model-checking problems for non-parametric temporal logic. PMITL turns out to be decidable and we show that, when parameter valuations give only non-singular sets, the considered problems are all

decidable, EXPSPACE-complete, and have the same complexity as in MITL. Moreover, we investigate the computational complexity of these problems for natural fragments of PMITL, and show that in meaningful fragments of the logic they are PSPACE-complete.

We also consider a remarkable problem expressed by queries where the values that each parameter may assume are either existentially or universally quantified. We solve this problem in several cases and we propose an algorithm in EXPSPACE.

Another interesting application of the temporal logic is when it is used to express specification of concurrent programs, where programs and properties are formalized as regular languages of infinite words. In this case, the verification problem (whether the program satisfies the specification) corresponds to solve the language inclusion problem.

In the second part of this thesis we consider the Synthesis problem for real-time systems, investigating the applicability of automata constructions that avoid determinization for solving the language inclusion problem and the realizability problem for real-time logics. Since Safra's determinization procedure is difficult to implement, we present Safraless algorithms for automata on infinite timed words.

To my father

”**Stay Hungry. Stay Foolish.**”

Steve Jobs

Acknowledgements

The work of this thesis would not have been possible without the guidance, encouragement, and support of many people.

First, I would like to thank my advisor, Prof. Margherita Napoli, for providing support and helpful comments whenever I needed it. I have particulary appreciated her guidance and her kindness.

I also thank Prof. Salvatore La Torre for fruitful discussions, support and precious suggestions.

I would like to thank Prof. Jean-François Raskin that made possible a productive visit in the Formal Methods and Verification Group at the Université libre de Bruxelles. It has been a great pleasure for me to work closely with him and with Gilles Geeraerts and Nathalie Sznajder. I am grateful to them for the excellent working atmosphere.

There are several friends who I should thank. First of all, I would like to thank my dear friend Eliana for her encouragements and suggestions. Our friendship has grown day to day and now it has a particular place in my heart.

Special thanks go to my colleagues and friends met in Bruxelles, above all my ”twin” Mahsa, Sara and Eliseo, that contributed in making my experience great. I will never forget their humor and friendship.

It would be too long to cite all of my friends and colleagues of Salerno but I would like to particularly thank Teresa, Anna, Sara, Carla, Rosaria, Roberto, my PhD mates and Raffaele for his true friendship.

Last but not least, I would like to thank my family, in particular my parents, and my boyfriend Federico, for their patience, love and for supporting me in every situation of my life.

Contents

List of Figures

Chapter 1

Introduction

In this thesis we concentrate our attention on Parametric and Real-Time Systems, studying some fundamental aspects of these systems.

In particular, we consider interesting decision problems regarding the formalisms that may underlie the specification and analysis of parametric and real-time systems. Let us first give a brief introduction of the context in Section 1.1, and recall the Model Checking problem (Section 1.2) and the Synthesis Problem (Section 1.3). Let us also point out the motivations (Section 1.4) of our research, and give a short description of the achieved results (Section 1.5).

1.1 System Verification

Currently, automated systems are present everywhere. Digital controllers are used to supervise critical functions of cars, airplanes, and industrial plants. Moreover, digital switching technology has replaced analog components in the telecommunication industry and security protocols enable e-commerce applications and privacy. When we consider safety critical applications, where important investments or even human lives are at risk, quality assurance for the underlying hardware and software components becomes paramount, and this requires formal models that describe the relevant part of the systems at an adequate level of abstraction [Merz, 2000].

For that reason, the development of formal methods for the design and analysis of safety-critical systems is an active area of computer science research.

The conventional approach to testing the correctness of a system involves simulation on some test cases, but, though this method is useful, it often results to be limited and so it is quite inadequate for developing bug free complex concurrent systems [Alur, 1991]. Another approach to assure correctness is to employ *automatic verification* methods.

Regarding this approach, a verification formalism comprises of: $a)$ a formal semantics which assigns mathematical meanings to system components, $b)$ a language for describing the essential characteristics of the system components with constructs for combining them, $c)$ a specification language for expressing the correctness requirements and $d)$ a verification algorithm used to check if the correctness criteria are fulfilled in every possible execution of the system [Alur, 1991] .

The systems we are focusing on are assumed to maintain an ongoing interaction with their environment (e.g., the controlled system or other components of a communication network) and, moreover, they are finite-state and real-time.

The essential characteristics of those reactive real-time systems [Alur, 1991] are:

- *Finite-state*: we consider that the system can be in one of the finitely many discrete states. This can be an useful abstraction in many cases if we focus only on the control aspect of the system, ignoring the computational aspect. State transitions are triggered by events which are instantaneous.

- *Reactive*: the system has a constant interaction with the environment reacting to stimuli. In this case the interest is for the ongoing behavior over time. This approach differs from the traditional "transformational" view of the programs where the functional relationship between the input state and the output state defines the meaning of a program.

- *Concurrent*: concurrency is a property of systems in which several computations are executing simultaneously, and potentially interacting with each other. The system comprises of a collection of components operating concurrently and communicating with each other.

- *Real-time*: the correct behavior of the system depends crucially on the time at which external events happen and not just on the ordering of events. This is obviously the case when the system needs to meet *hard* real time deadlines: the

system needs to respond to a stimulus within a certain fixed time bound. Also there are cases when the logical correctness of the system depends on the lengths of various delays.

Real-time systems are used in safety critical applications such as flight controller or controllers for nuclear plants, so failures in such systems can be very expensive and even life threatening. We notice that since there are complicate timing relationships, real time systems are quite hard to design. Consequently, in order to simplify the modeling and specification processes there is a great demand for formal methods applicable to real-time systems. Formal methods are gaining popularity as a way to establish system correctness mathematically, by proving that a formal model of the system satisfies a given property.

Moreover, another characteristic of real-time systems is the following: the behavior of those systems can be either influenced or not by the interaction with the environment. For this reason, in computer system design, there is a distinction between *closed* and *open* systems. A closed system is a system where both program and user work together to find the required output and whose behavior is completely determined by the state of the system. An open system is a system that interacts with its environment and whose behavior depends on this interaction; it assumes a hostile environment.

1.2 Model Checking

Model Checking is an automatic technique for verifying finite state concurrent systems. Formally, the *Model Checking problem* can be stated as follows: let M be the model of the system, and let ϕ be the requirement, that represents a specification. We want to find all states s of M such that s satisfies the given specification. Model checking is used to determine if M is a model for ϕ.

It has a number of advantages over traditional approaches that are based on simulation, testing or deductive reasoning. Also, if the design contains an error, model checking will produce a counterexample that can be used to pinpoint the source of the error. The method has been used successfully in practice to verify real industrial designs, and companies are beginning to market commercial model checkers. For example, it has been applied to the verification of sequential circuit designs and communication

protocols [Clarke et al., 2000].

The process of Model Checking consists of three main tasks:

1. Modelling: the first task is to convert a design into a formalism accepted by a model checking tool.

2. Specification: this task is to state the properties that the design must satisfy. The specification is usually given in some logical formalism. For hardware and software systems, it is common to use temporal logic, which can assert how the behavior of the system evolves over time.

3. Verification: ideally the verification is completely automatic. However, in practice it often involves human assistance for the analysis of the verification results. In case of a negative result, the user is often provided with an error trace. This can be used as a counterexample for the checked property and can help the designer in tracking down where the error occurred.

Model checking is being adopted as a standard procedure for the quality assurance of reactive systems because it has been proven cost-effective and integrates well with conventional design methods.

The inputs to a Model checker are a (usually finite-state) description of the system to be analyzed and a number of properties, often expressed as formulas of temporal logic, that are expected to hold of the system [Merz, 2000]. The model checker either confirms that the properties hold or reports that they are violated. In the latter case, it provides a counterexample: a run that violates the property. Such a run can provide valuable feedback and points to design errors.

Another interesting application of model checking is *Software model checking* [Holzmann and Smith, 1999] that is the algorithmic analysis of programs to prove properties of their executions. In this case model checkers can be used in the validation process: ensure that the abstract model adequately reflects the behavior of the concrete system in order for the properties of interest to be established or falsified. Indeed, it is possible to perform checks in order to ensure that certain runs are possible or that the model is free of deadlocks.

Compared to other verification techniques, such as automated theorem proving or proof checking, model checking has different advantages, like the following [Clarke,

2008]:

- *The correctness proofs are not necessary*. The user of a model checker does not need to construct manually a correctness proof. In principle, all that is necessary for the user is to enter a description of the system to be verified, that may be a circuit or a program, and the specification to be checked. The checking process is automatic.

- *The process is fast*. In practice, model checking is fast compared to other rigorous methods such as the use of a proof checker, which may require months while the user is working interactively with the proof checker.

- *Diagnostic counterexamples*. If the specification is not satisfied, the model checker will produce a counterexample execution trace that shows why the specification does not hold. The counterexamples are invaluable in debugging complex systems. Some people use model checking just for this feature.

- *Partial specifications are possible*. It is unnecessary to completely specify the system before beginning to model check properties. Thus, model checking can be used during the design of a complex system. The user does not have to wait until the design phase is complete.

- *Formalism for concurrent system specifications*. Temporal logics can easily express many of the properties that are needed for reasoning about concurrent systems. This is important because the reason some concurrency property holds is often quite subtle, and it is difficult to verify all possible cases manually.

1.3 The Synthesis Problem

Given a specification, the *Synthesis* (or *Realizability*) problem is the automatic construction of a design that is guaranteed to be correct. If a system has been specified precisely, then it should be possible to generate the design automatically, avoiding the costs of separately developing a possibly incorrect system.

Synthesis aims to transform a specification into a system that is guaranteed to satisfy the specification. The theory behind synthesis of reactive systems is well

established and goes back to Church [1962], who introduced the Synthesis Problem using different fragments of restricted recursive arithmetic (S1S) as specification [Friedman, 1957].

In the design of real-time reactive systems, it is usually difficult to construct manually a precise model of the reactive system; moreover, the environment may be only partially known, especially in the early stages of development. Therefore, it is natural to consider the problem of the automatic synthesis of a behavior policy for the reactive system that would be correct by construction with respect to the specification. As we have seen in Section 1.2, the goal of model checking is to verify that a given system matches its specification without error. Further to model checking, we may wish to automatically construct a functional correct system from a behavioral description of the system; this is the idea of synthesis.

The Synthesis approach presents several advantages [Jobstmann, 2007], like the following:

- *The process is easy*: we only have to give a list of desired behaviors and a synthesis tool comes up with a state model that takes all demanded properties into account. For systems that have no further constraints (e.g., on timing or space consumption) a synthesis tool would completely avoid hand-coding.

- *Use of synthesis for the construction of prototypes*: with the synthesis process there is the possibility to construct rapid prototypes from specification.These functional prototypes could be used for early test integration and would allow to ”simulate the specification”. Most developers rely on simulation to check if the constructed system meets their intents. Synthesis is an extremely good way to validate and debug a specification.

- *Application of the synthesis in several branches of engineering*: automatic construction of correct systems has attracted much attention in the engineering field. For example, in Very Large Scale Integrated (VLSI) circuits, where finite-state machines constitute the basic building blocks of such circuits, the synthesis problem asks, given a specification characterizing the set of permissible implementation, for the construction of a finite-state machine M allowed by the specification such that M satisfies some optimality criteria [De Micheli, 1994].

In the open setting, the synthesis problem is usually formalized as a two-players game [Doyen et al., 2009], in which Player 1 controls the execution of the system, and Player 2 controls the execution of environment. The specification is encoded as the winning condition for Player 1 in the game. Roughly speaking, the behaviors of Player 1 represent all possible models for the system, and computing a winning strategy for Player 1 amounts to selecting one model which is guaranteed to be correct whatever the environment does.

1.4 Motivations

From the main characteristics of real-time systems, it is easy to see that there is a need for formal methods.

As we have seen in Section 1.2, the standard formalism used to specify the wished behaviour of a real-time system is the *temporal logic*. Its use as a specification language was first suggested by Pnueli [Pnueli, 1977] who proposed the propositional linear temporal logic (LTL). This logic presents natural operators to express temporal requests on the time ordering of occurrences of events, such as "always", "eventually", "until", and "next".

Traditional temporal logics, such as LTL, allow only qualitative assertions about the temporal ordering of events. In several circumstances, for assessing the efficiency and practicality of the system being modeled, it may be useful to have additional quantitative guarantees (e.g. time differences between given events).

For this reason, Alur et al. [1996] introduced the Metric Interval Temporal Logic (MITL) that extends LTL with a real-time semantics where changes of truth values happen according to a splitting of the line of non-negative reals into intervals. Syntactically, MITL augments the temporal operators of LTL with a subscript which expresses an interval of interest for the expressed property. Thus, properties such as "every time an a occurs then a b must occur within time $t \in [3, 5]$" become expressible.

However, even with quantitative temporal logics, model checking still yields only a "yes/no" answer, so we would actually like to find out what quantitative bounds can be placed on the logic operators. Consequently, it arises the need of introducing parameters, considered as variables that abstract time values, in real-time formalisms, such as in timed models and temporal logic formulas.

When the designer cannot fully characterize the system or when the system may be affected by unknown features of the environment, it can be useful to introduce parameters in timed models in place of unknown or undefined values. Parameters can be used to specify properties of a system and to model the system itself. The first parametric model is the Parametric Timed Automaton (PTA) [Alur et al., 1993c] introduced by Alur, Henzinger and Vardi; it allows parameters in clock constraints.

Unfortunately, the parametric verification problem is very difficult. Indeed the emptiness problem for parametric timed automata, whose solution is used for the verification of parametric real-time specifications, reveals that the number of clocks in a parametric timed automaton is critical to the decidability of the problem [Alur et al., 1993c]. For this reason it is better to consider a different parametric model.

The main motivation for introducing the parameters in a logic formalisms is to allow a thorough analysis of the possible values of the numeric constants used in the design. The possibility of using such parametric constants is of great appeal mostly in the early stages of a design when, due to the scarce information on the system, the exact value of these constants is hard, or even impossible, to determine. In these regards, it is useful to be able to characterize the domains of the parameter valuations that make the considered analysis true.

The use of parametric constants has been advocated by many authors as a support to designers (see for example [Alur et al., 1993c, 2001; Bruyère and Raskin, 2003; Courcoubetis and Yannakakis, 1992; Emerson and Trefler, 1999; Hune et al., 2002; Wang, 1996]). Unfortunately, the unrestricted use of parameters leads to undecidability results [Alur et al., 1993c, 2001] and characterizing the domains of the parameter valuations is possible only when we restrict to use only parameterized operators of just one polarity. The reason is essentially that if we can characterize such domains in an algorithmically usable format for the full logic, we would be able to answer the decision problems also when parameter valuations evaluating parameterized intervals to singular sets are allowed.

One of the crucial points concerning parametric systems is to explicitly compute the set of all the valuations that make a parameterized formula satisfiable (or valid). In [Alur et al., 2001], this problem has been solved for a parametric logic (PLTL) with a discrete semantics when all the operators have the same polarity, using an algorithm that takes double-exponential time in the number of parameters. It is very interesting

to seek algorithms which get more information on this set of parameter valuations also when parameters of both polarities are allowed.

Another interesting application of the temporal logic is when it is used to express specification of concurrent programs, where programs and properties are formalized as regular languages of infinite words. Any regular language of infinite words is accepted by a nondeterministic Büchi automaton. In practice, if we denote with A the non deterministic Büchi automaton that formalizes the program, and with B the non deterministic Büchi automaton that formalizes the specification, the verification problem (whether the program satisfies the specification) corresponds to test if $L(A) \subseteq L(B)$ that is if $L(A) \cap L(\neg B) = \varnothing$. With this procedure, $\neg B$ is obtained by determinization of B. Nevertheless, currently there is no practical algorithms to solve this language inclusion problem. The usual approach through explicit complementation is difficult. It is interesting to focus on the automata-theoretic approach that separates the logical and the combinatorial aspects of reasoning about programs.

Safra introduced an optimal determinization construction [Safra, 1988], but this procedure is difficult to implement even in the context of untimed languages. The implementation of Safra's algorithm [Tasiran et al., 1995] has to cope with the involved structure of the states in the complementary automaton. The lack of a simple implementation is not due to a lack of need. As a consequence, recent research efforts have investigated alternative decision procedures [Filiot et al., 2009; Kupferman and Vardi, 2001, 2005b; Schewe and Finkbeiner, 2007] that avoid the use of this construction.

All the solutions proposed do not consider timed specification that are useful to express properties of real-time systems.

It could be interesting try to adapt the techniques of Kupferman and Vardi [2001] for alternating event-clock automata, in order to solve the universality and language inclusion problems for that class of timed automata without resorting to the Safra's construction.

If we are considering open systems, an interesting goal is to generalize the ideas introduced for the language inclusion problem to solve the synthesis problem for a fragment of the Event Clocks Logic (ECL) called $\mathsf{LTL}_{\triangleleft}$ [Doyen et al., 2009]. We focus on the fragment $\mathsf{LTL}_{\triangleleft}$ for which the realizability problem is decidable, rather then the full ECL for which it is undecidable. To formalize the synthesis problem, we may consider it in terms of games.

Furthermore, a significant direction is the introduction of parameters in the open setting, with the analysis of the synthesis problem for parametric real-time formalisms. Therefore it is useful to consider the study of the techniques that may be used to solve parametric games.

1.5 Contributions

The main contribution of this thesis is to analyze the parametric and real-time systems by considering the verification and synthesis problems. To be clearer, we can gather the contributions into two main categories:

1. In the first part of this thesis we focus on closed real-time systems and we consider the decision problems, such as the Model Checking problem defined in Section 1.2.

2. In the second part of this thesis we focus on open real-time systems, assuming the presence of an hostile environment, and we consider decision problems, such as the Synthesis Problem, defined in Section 1.3.

More precisely, regarding the part (1), we introduce the *Parametric Metric Interval Temporal Logic*, an extension of MITL with parametric constants, i.e., we allow the intervals in the subscripts of the temporal operators to have as an endpoint a *parametric expression* of the form $c + x$, for a parameter x and a constant c. Therefore, typical time properties which are expressible in MITL can now be analyzed by varying the scope of the temporal operators depending on the values of the parameters. The logic PMITL adds to MITL in terms of expressiveness, in the sense that some properties can be expressed in PMITL but not in MITL. We show that the satisfiability, validity and model-checking problems are all EXPSPACE-complete within this formalism.

In order to obtain decidability results, we use a particular model and we impose some restrictions on the parameters. Indeed, in the first case we introduce an interesting class of PTA: *lower bound/upper bound* (L/U) PTA *automata*, in which each parameter occurs either as a lower bound L or as an upper bound U in the timing constraints. We have decided to use this class of automata because, despite the apparent limitation, the model is still interesting in practice. In the second case, we define

PMITL by imposing the following restrictions on the parameters. First, the sets of parameters L and U have to be disjoint. Second, we force each interval to have at most one parameter, either in the left or in the right end-point. Third, we define admissibility for parameter valuations such that a parameterized interval cannot be evaluated neither as an empty nor a singular set. In particular, we show that relaxing any of the first two restrictions leads to undecidability.

To get the whole picture of the PMITL analysis, we focus on the study of the computational complexity of natural syntactic fragments of PMITL, showing that in meaningful fragments of the logic the considered decision problems are PSPACE-complete. Moreover, we make a progress in the analysis of characterizing the space of the fulfilling parameter valuations to explicitly compute the set of all the valuations that make a parameterized formula satisfiable (or valid), also when parameters of both polarities are allowed. More precisely, we study a general decision problem which gives more information on this space than simply considering the emptiness and universality problems [Di Giampaolo et al., 2010b]. We solve this problem in several cases and exhibit an algorithm in EXPSPACE.

Regarding the group (2), we introduce an automata-theoretic approach that avoid determinization for solving the language inclusion problem and the synthesis problem for real-time logics. In particular, we focus on timed specification, considering extensions of Safraless algorithms proposed in the literature for automata on infinite untimed words to the case of automata on infinite timed words. The principal contribution in this part of the thesis is to show that the techniques of Kupferman and Vardi [2001] can be adapted to alternating event-clock automata. Those procedures can then be used to complement nondeterministic event-clock automata with Büchi acceptance conditions and this in turn leads to algorithms for solving the universality and language inclusion problems for that class of timed automata without resorting to the Safra's construction [Di Giampaolo et al., 2010a].

Furthermore, we generalize the ideas, introduced for the language inclusion problem, to solve the synthesis problem for a fragment of the Event Clocks Logic (ECL), called $\mathsf{LTL}_{\triangleleft}$ [Doyen et al., 2009]. We focus on the fragment $\mathsf{LTL}_{\triangleleft}$ for which the realizability problem is decidable [Doyen et al., 2009], because $\mathsf{LTL}_{\triangleleft}$ formulas can express interesting properties of reactive systems. In this thesis, we show how to reduce the original problem to a timed safety game problem, which can be solved in practice

thanks to tools such as UPPAAL TIGA TiGa [Behrmann et al., 2007].

1.6 Organization of this thesis

The thesis is organized as follows.

Chapter 2 contains the basic definitions to represent real-time behaviour starting from the trace semantics. This chapter recall the definitions and properties of formalisms used to define temporal specifications of real-time systems. These formalisms are used to represent models of the systems, such as timed automata, Büchi automata, event clock automata and to represent specification properties of the system such as linear temporal logic, noted LTL, metric temporal logic, noted MTL and metric interval temporal logic, noted MITL. We recall the main results that are known about those formalism.

Chapter 3 focuses on the introduction of parameters in real-time formalisms, explaining the importance of the use of parameters as variables that abstract time values. Parameters can be introduced in timed models, as in the case of parametric timed automata, noted PTA, and an interesting class of PTA: *lower bound/upper bound* (L/U) PTA. Parameters can also be introduced in temporal logics. An example is given by the parametric temporal logic (PLTL), in which temporal operators can be subscripted, together with a direction, by a variable ranging over the natural numbers, and the logic PROMPT-LTL with a new temporal operator that is used for specifying eventualities with a bounded wait time. We recall the main results that are known about those formalism for problems such as satisfiability and model checking. Finally, we introduce the parametric notation used in this thesis, as parameterized intervals, expressions and a model of parametric timed automata.

Chapter 4 introduces the logic PMITL along with the related decision problems, and gives a comparison with MITL. It is showed the concept of polarity of parameterized temporal operators. Moreover, it is given an example of a model of the SPSMALL memory, a commercial product of STMicroeletronics. Then, there are shown interesting properties that can be expressed using our formalism. Decidability results regarding the satisfiability, validity, and model-checking problems are proved. A central step in our argument is a translation to the emptiness and the universality problems for Büchi L/U automata.

Chapter 5 is dedicated to the study of the computational complexity of natural syntactic fragments of PMITL: $P_{0,\infty}MITL_{0,\infty}$, $PMITL_{0,\infty}$, $PMITL_{\Diamond}$ and $PMITL_{\Box}$.

On the positive side, it is proved that some of the considered problems are PSPACE for $PMITL_{\Diamond}$ and $PMITL_{\Box}$. These fragments are quite expressive (for example it is possible to express properties such as the parameterized response property) and, to the best of our knowledge, the union of these fragments captures the most general known formulation of parametric constraints in PMITL with the considered decision problems in PSPACE.

Moreover, two decidable generalizations of our results are discussed. In the first one, our logic is extended by allowing parametric expressions which are *linear expressions* of the parameters. The other generalization is a more general formulation of the considered decision problems. Decision problems are expressed as queries where each parameter is quantified either existentially or universally. Finally, it is shown that relaxing the restrictions imposed over the parameterized intervals leads to undecidability.

Chapter 6 focuses on the applicability of automata constructions that avoid determinization for solving the language inclusion problem. First, it deals with the Safra's procedure used for the construction of a deterministic ω-automaton equivalent to a non deterministic ω-automaton. Since Safra-based determinization is difficult to implement, even in the context of untimed languages, alternative decision procedures, recently investigated, are presented in order to avoid the use of this construction, such the "rank construction" of Kupferman and Vardi. In this chapter we investigate extensions of those techniques to timed languages expressed by (alternating) event-clock automata. That is, given an alternating event-clock automaton with co-Büchi acceptance condition A, we show how to construct, in quadratic time, an alternating event-clock automaton with Büchi acceptance condition B that accepts the same language as A. From that alternating event-clock automaton B, we show how to construct in exponential time a nondeterministic event-clock automaton C with Büchi acceptance condition such that accepts the same language as B and A. Those procedures then can be used to complement nondeterministic event-clock automata with Büchi acceptance conditions, this in turn leads to algorithms for solving the universality and language inclusion problems for that class of timed automata without resorting to the Safra's construction.

Chapter 7 introduces the realizability problem for real time logics, starting form

the definition of LTL synthesis with the classical solutions obtained with Safra determinization and defining Safraless approaches for LTL synthesis. In this chapter the ideas about Safraless procedures are generalized in order to solve the realizability problem for a fragment of the Event Clocks Logic called $\mathsf{LTL}_{\triangleleft}$. For each formula of this logic, it is possible to construct, in exponential time, a universal event-clock automaton with co-Büchi acceptance condition that accepts the set of timed words that the formula defines. Then, we show that the co-Büchi acceptance condition can be strengthened into a condition that asks that all runs of the automaton visit less than $K \in \mathbb{N}$ times the set of accepting locations. This allows to reduce the realizability problem for $\mathsf{LTL}_{\triangleleft}$ to the realizability problem for universal K-co-Büchi event-clock automata. Those are easily determinizable and this reduces the original problem to a timed safety game problem. Moreover we show that this timed safety game problem can be solved using the model checking tool UPPAAL TIGA illustrating this on a simple example.

Chapter 8 concludes this thesis giving some directions for future works.

Chapter 2

Models and Specifications of the System

In this chapter we introduce the basic definitions for the representation of real-time behavior. We consider formalisms used to represent models and to define temporal properties of real-time systems, recalling all the results that are known about those formalisms.

2.1 Models of the System

Real-time systems can be broadly classified in two categories: *distributed systems* whose subcomponents are spatially separated and *concurrent systems* that share resources such as processors and memories [Merz, 2000]. Distributed systems communicate by message passing, whereas concurrent systems may use shared variables. Concurrent processes may share a common clock and execute in lock-step, as for time-synchronous systems, typical for hardware verification problems, or operate asynchronously, sharing a common processor. In the latter case, one should ensure processes that could execute are eventually scheduled for execution. A common framework for the representation of these different kinds of systems is provided by the concept of *transition systems*.

2.1.1 Transition systems

Given a set of atomic propositions AP, a transition system $\mathcal{T}$ over AP is a four tuple $\mathcal{T} = (Q, Q_0, \delta, \lambda)$, where:

- Q is a finite set of states
- $Q_0 \subseteq Q$ is the set of initial states
- $\delta \subseteq Q \times Q$ is a total transition relation that is, we require that for every state $q \in Q$ there exist a state $q' \in Q$ such that $(q, q') \in \delta$.
- $\lambda : Q \to 2^{AP}$ is a function that labels each state with the set of atomic propositions true in that state.

A run of $\mathcal{T}$ is an infinite sequence $\rho = q_0 q_1 \ldots$ of states $q_i \in Q$ such that $q_0 \in Q_0$ and for all $i \in \mathbb{N}$, $(q_i, q_{i+1}) \in \delta$.

A transition system specifies the allowed evolutions of the system: starting from some initial state, the system evolves by performing actions that take the system to a new state. Slightly different definitions of transition systems abound in the literature. An example is given by the term Kripke structure, in honor of the logician Saul A. Kripke who used transition systems to define the semantics of modal logics [Kripke, 1963].

In practice, reactive systems are described using modelling languages, including (pseudo) programming languages such as PROMELA, but also process algebras or Petri nets. The operational semantics of these formalisms is conveniently defined in terms of transition systems. However, the transition system that corresponds to such a description is typically of size exponential in the length of the description. For example, the state space of a shared-variable program is the product of the variable domains. Modelling languages and their associated model checkers are usually optimized for particular kinds of systems such as synchronous shared-variable programs or asynchronous communication protocols. In particular, for systems composed of several processes it is advantageous to exploit the process structure and avoid the explicit construction of a single transition system that represents the joint behavior of processes.

Given a transition system $\mathcal{T}$, it is usual to ask questions such as the following:

- Are any "undesired" states reachable in $\mathcal{T}$, such as states that represent a deadlock, a violation of mutual exclusion etc.?
- Are there runs of $\mathcal{T}$ such that, from some point onwards, some "desired" state is never reached or some action never executed? Such runs may represent livelocks where, for example, some process is prevented from entering its critical section, although other components of the system may still make progress.
- Is some initial system state of $\mathcal{T}$ reachable from every state? In other words, can the system be reset?

Temporal logic [Emerson, 1990; Kröger, 1987; Manna and Pnueli, 1992b, 1995; Stirling, 1992] is a convenient language to formally express such properties.

2.1.2 Trace semantics

We consider the notion of trace semantics in order to introduce time in linear trace semantics for representing concurrent processes [Alur, 1991].

In trace semantics, it is usual to associate a set of observable events with each process, and model the process by the set of all its traces. A *trace* is a linear sequence of events that may be observed when the process runs.

For example, an event may denote an assignment of a value to a variable, or pressing a button on the control panel, or arrival of a message. All events are assumed to occur instantaneously. Actions with duration are modeled using events marking the beginning and the end of the action. We consider only infinite sequences, which model nonterminating interaction of reactive systems with their environments.

Words Formally, an alphabet Σ is a finite set of letters. A *finite (resp. infinite) word* w over an alphabet Σ is a finite (resp. infinite) sequence of letters from Σ. We denote respectively by Σ^* and Σ^ω the sets of all finite and infinite words on Σ. We denote by ε the empty word, and by $|w|$ the length the word w (which is equal to ∞ when w is infinite). Given a set of atomic proposition AP, a trace $\alpha = \alpha_0\alpha_1\ldots$ is an infinite word, where $\alpha_i \in 2^{AP}$.

Timed traces An untimed process models the sequencing of events but not the actual times at which the events occur. Timing can be added to a trace by coupling it with a sequence of time values.

We assume that these values are chosen from a domain $\mathcal{D}$ with linear order $\leq$. Different choices for $\mathcal{D}$ will lead to different ways of modeling the behavior. A time sequence $\tau = \tau_0\tau_1\ldots$ is an infinite sequence of time values $\tau_i \in \mathcal{D}$ with $\tau_i > 0$, satisfying the following constraints:

- Monotonicity: τ increases strictly monotonically, that is $\tau_i < \tau_{i+1}$ for all $i \geq 1$.
- Progress: for all $t \in \mathcal{D}$ there is some $i \geq 1$ such that $\tau_i > t$

A *finite (resp. infinite) timed trace (or word)* over an alphabet Σ is a pair $\theta = (w, \tau)$ where w is a finite (resp. infinite) word over Σ, and $\tau = \tau_0\tau_1\ldots\tau_{|w|-1}$ is a finite (resp. infinite) sequence of length $|w|$ of positive real values (the time stamps) such that $\tau_i \leq \tau_{i+1}$ for all $0 \leq i \leq |w| - 1$ (resp. for all $i \geq 0$). We let $|(w, \tau)| = |w|$ denote the length of (w, τ).

For a timed trace (α, τ), over a a set of atomic propositions AP, each τ_i gives the time at which the proposition α_i occurs.

An infinite timed word $\theta = (w, \tau)$ is *diverging* if for all $t \in \mathbb{R}^{\geq 0}$, there exists a position $i \in \mathbb{N}$ such that $\tau_i \geq t$. We denote respectively by $\mathsf{T}\Sigma^*$, $\mathsf{T}\Sigma^\omega$ and $\mathsf{T}\Sigma^\omega_{\mathrm{td}}$ the sets of all finite, infinite and infinite diverging timed words on Σ. In the sequel, it is often convenient to denote an (infinite) timed word (w, τ) by the sequence $(w_0, \tau_0)(w_1, \tau_1)\ldots$.

We proceed similarly for finite timed words. Since we are interested mainly in *infinite* timed words, we often refer to them simply as *timed words*.

We have a *discrete*-time model if we choose the domain $\mathcal{D}$ to be the set of natural numbers $\mathbb{N}$. In this model events can happen only at the integer time values. This describes the behavior of synchronous systems, where all components are driven by a common global clock. The duration between the successive clock ticks is chosen as the time unit. The discrete-time model is the traditional model for synchronous hardware. The advantage of this model is its simplicity. In fact, timed traces are not even necessary to model the behavior.

Choosing $\mathcal{D}$ to be the set of real numbers $\mathbb{R}$ gives the *dense*-time model. In this model, we assume that events happen at arbitrary points in time over the real line, and with each event we associate its real valued time of occurrence. As it turns out, with

regards to complexity and expressiveness issues, the crucial aspect of the underlying domain is its denseness, that is the property that between every two time values there is a third one, and not its continuity. The dense-time model is a natural model for asynchronous systems. It allows events to happen arbitrarily close to each other; that is, there is no lower bound on the separation between events. This is a desirable feature for representing two causally independent events in an asynchronous system.

Since the dense-time model admits the possibility of an unbounded number of events in an interval of finite time length, some problems related to the verification of finite state systems turn out to be, unlike the other models, undecidable. For example, the language inclusion problem for timed automata is undecidable; if we had chosen one of the discrete models, the problem would have been solvable.

Timed sequence over intervals We consider non-empty intervals (convex sets) of non-negative real numbers. We use the standard notation $[a, b]$, $]a, b[$, $[a, b[$, and $]a, b]$ to denote respectively the closed, open, left-closed/right-open and left-open/right-closed intervals with end-points a and b. When we do not need to specify if an end-point is included or not in an interval, we simply use parentheses: for example, we denote with (a, b) any of the possible intervals with end-points a and b. A *time interval* I is an interval (a, b) such that $0 \leq a \leq b$, and $a < b$ if I is not closed. A closed time interval $I = [a, a]$ is called *singular*. Given an interval $I = (a, b)$ and $t \geq -a$, with $I + t$ we denote the interval $(a + t, b + t)$ such that I is left-closed (resp. right-closed) iff $I + t$ is left-closed (resp. right-closed).

An *interval sequence* is an infinite sequence $I_0, I_1 \ldots$ of time intervals such that:

- for all i, $I_i \cap I_{i+1} = \varnothing$ and, denoting $I_i = (a_i, b_i)$, $a_{i+1} = b_i$ holds (*along the time line I_{i+1} follows I_i*);
- each real number $t \geq 0$ belongs to some interval I_i (*the sequence of intervals covers the reals*).

We fix a set of atomic propositions AP. A *timed sequence* over AP is an infinite sequence $\alpha = (\alpha_0, I_0)(\alpha_1, I_1) \ldots$ such that $\alpha_i \in 2^{AP}$, for all i, and $I_0, I_1 \ldots$ is an interval sequence. For each $t \geq 0$, $\alpha(t)$ denotes the unique α_i such that $t \in I_i$.

2.1.3 ω- automata

An ω language consists of infinite words. Thus an ω language over an alphabet Σ is a subset of Σ^ω. The ω- automata provide a finite representation for certain types of ω languages. An ω automaton is essentially the same as a nondeterministic finite-state automaton, but with the acceptance condition modified suitably so as to handle infinite input words.

The theory of automata over infinite words and trees was initiated by Büchi [1962], Muller [1963], and Rabin [1969].

Different types of ω automata are defined according to the acceptance condition. An ω language is called ω-regular if it is accepted by some Büchi automaton. We present some of its basic elements.

A Büchi automaton $\mathcal{B} = \langle Q, Q^0, \Delta, F \rangle$ over an alphabet Σ is given by a finite set Q of locations, a non-empty set $Q^0 \subseteq Q$ of initial locations, a transition relation $\Delta \subseteq Q \times \Sigma \times Q$, and a set $F \subseteq Q$ of accepting locations.

A run of $\mathcal{B}$ over an ω-word $w = a_0 a_1 \ldots \in \Sigma^\omega$ is an infinite sequence $\rho = q_0 q_1 \ldots$ of locations $q_i \in Q$ such that $q_0 \in Q^0$ and $(q_i, a_i, q_{i+1}) \in \Delta$ holds for all $i \in \mathbb{N}$. The run ρ is accepting iff there exists some $q \in F$ such that $q_i = q$ holds for infinitely many $i \in \mathbb{N}$.

The language $L(\mathcal{B}) \subseteq \Sigma^\omega$ is the set of ω-words for which there exists some accepting run ρ of $\mathcal{B}$. A language $L(\mathcal{B}) \subseteq \Sigma^\omega$ is called ω-regular iff $L = L(\mathcal{B})$ for some Büchi automaton $\mathcal{B}$.

Büchi automata are presented just as ordinary (non-deterministic) finite automata over finite words [Hopcroft and Ullman, 1979].

The notion of "final locations", which obviously does not apply to ω-words, is replaced by the requirement that a run passes infinitely often through an accepting location.

Many properties of classical finite automata carry over to Büchi automata. For example, the emptiness problem is decidable. Unlike the case of standard finite automata, deterministic Büchi automata are strictly weaker than non-deterministic ones.

It is therefore impossible to prove closure of the class of ω-regular languages under complement in the standard way (first construct a deterministic Büchi automaton equivalent to the initial one, then complement the set of accepting locations).

Nevertheless, Büchi [1962] has shown that the complement of an ω-regular language is again ω-regular. His proof relied on combinatorial arguments (Ramsey's theorem) and was non-constructive. A succession of papers has replaced this argument with explicit constructions, culminating in the following result due to Safra [1988] of essentially optimal complexity, which we recall:

Proposition 1 *For a Büchi automaton $\mathcal{B}$ with n locations over alphabet Σ there is a Büchi automaton $\neg\mathcal{B}$ with $2^{O(nlogn)}$ locations such that $L(\neg\mathcal{B}) = \Sigma^\omega \setminus L(\mathcal{B})$.*

2.1.4 Timed automata

A timed automaton is essentially a finite automaton (that is a graph containing a finite set of nodes or locations and a finite set of labeled edges) extended with real-valued variables. Such an automaton may be considered as an abstract model of a timed system. The variables model the logical clocks in the system, that are initialized with zero when the system is started, and then increase synchronously with the same rate. Clock constraints i.e. guards on edges are used to restrict the behavior of the automaton. A transition represented by an edge can be taken when the clocks values satisfy the guard labeled on the edge. Clocks may be reset to zero when a transition is taken.

The model of timed automata was first introduced in 1990 by Alur and Dill [1994] as an automata-theoretic approach for describing and analyzing the behaviour of finite-state systems with real-valued clocks.

We briefly recall the definition. Given a finite set of clocks X, a *clock constraint* is a is a conjunctive formula of atomic constraints of the form $\xi \approx e$ where $\xi \in X$, $\approx \in \{<, \leq, \geq, >\}$, and e is a nonnegative rational number. We denote with $C(X)$ the set of all clock constraints over X.

A *clock interpretation* $\sigma : X \to \mathbb{R}_+$ assigns real values to each clock. A clock interpretation $\sigma + t$, for $t \in \mathbb{R}_+$, assigns $\sigma(\xi) + t$ to each $\xi \in X$. For $\gamma \subseteq X$, $\sigma[\gamma := 0]$ denotes the clock interpretation that assigns 0 to all clocks in γ, and $\sigma(\xi)$ to all the other clocks ξ.

A clock interpretation σ satisfy a clock constraint δ over X iff evaluating each clock of δ according to σ the resulting boolean expression holds true.

Now, we give the formal definition.

A *timed automaton* (TA) is a tuple $\mathcal{A} = \langle Q, Q^0, X, \Delta, \lambda \rangle$, where Q is a finite set of *locations*, $Q^0 \subseteq Q$ is the set of *initial locations*, X is a finite set of clocks, $\Delta \subseteq Q \times C(X) \times Q \times 2^X$ is a *transition relation*, and $\lambda : Q \to 2^{AP}$ is a function labeling each location with a set of atomic propositions.

At time τ_i, an edge (q, q', δ, γ) represents a transition from state q to state q'. The set δ is a clock constraint over X. The set $\gamma \subseteq X$ gives the clocks to be reset with this transition.

For locations $q_i \in Q$, clock interpretations σ_i, clock constraints $\delta_i \in C(X)$, clock sets $\gamma_i \subseteq X$, a *run* ρ of a TA $\mathcal{A}$, is an infinite sequence

$\xrightarrow{\tau_0} (q_0, \sigma_1) \xrightarrow[\delta_1,\gamma_1]{\tau_1} (q_1, \sigma_1) \xrightarrow[\delta_2,\gamma_2]{\tau_2} (q_2, \sigma_2) \xrightarrow[\delta_3,\gamma_3]{\tau_3} \ldots$, such that:

- $q_0 \in Q^0$, and $\sigma_0(\xi) = 0$, for each clock $\xi \in X$;
- for all $i \geq 0$, there in ad edge of the form $(q_i, q_{i+1}, \delta_i, \gamma_i) \in \Delta$, such that

  (1) $(\sigma_i + \tau_{i+1} - \tau_i) \models \delta_i$ and

  (2) $\sigma_{i+1} = [\gamma_i := 0](\sigma_i + \tau_{i+1} - \tau_i)$.

The timed sequence associated with ρ is $(\lambda(q_0), \tau_0)(\lambda(q_1), \tau_1)(\lambda(q_2), \tau_2) \ldots$.

2.1.5 Event clock automata

The theory of timed automata has a great importance because allows the solution of certain verification problems for real-time systems with finite control [Alur and Dill, 1994; Alur et al., 1993a, 1995, 1996; Henzinger et al., 1994b] and the solution of certain delay problems [Alur et al., 1993b; Courcoubetis and Yannakakis, 1992].

Moreover, based on this theory, it have been implemented several automatic tools, including Cospan [Alur and Kurshan, 1995], Kronos [Daws et al., 1995], and UPPAAL TIGA [Bengtsson et al., 1995].

However, the general verification problem (i.e., language inclusion) is undecidable for timed automata [Alur and Dill, 1994].

The main reason is because, unlike in the untimed case, the nondeterministic variety of timed automata is strictly more expressive than the deterministic variety.

Alur et al. [1994] obtain a determinizable class of timed automata by restricting the use of clocks and introducing *event-clock automata* (ECA). The clocks of an event-clock automaton have a fixed, predefined association with the symbols of the input alphabet as the alphabet symbols typically represent events. The event-recording clock of the input symbol a is a history variable whose value always equals the time of the last occurrence of a relative to the current time; the event-predicting clock of a is a prophecy variable whose value always equals the time of the next occurrence of a relative to the current time (if no such occurrence exists, then the clock value is undefined). Thus, unlike a timed automaton, an event-clock automaton does not control the reassignments of its clocks, and, at each input symbol, all clock values of the automaton are determined solely by the input word. This property allows the determinization of event-clock automata, which, in turn, leads to a complementation procedure. Indeed, the class ECA of event-clock automata is closed under all boolean operations (timed automata are not closed under complement), and the language-inclusion problem is decidable and PSPACE-complete for event-clock automata.

The class of event-clock automata is sufficiently expressive to model real-time systems with finite control, and to specify common real-time requirements. For instance, the hard real-time requirements that "every request is followed by a response within 3 seconds" and that "every two consecutive requests are separated by at least 5 seconds" can be expressed using event-clock automata. In fact, the authors argue that automata that contain only event-recording clocks, which are called *event-recording automata* (PastECA) are a suitable abstract model for real-time systems by proving that event-recording automata are as powerful as another popular model for real-time computation, timed transition systems [Henzinger et al., 1994a].

A timed transition system associates with each transition a lower bound and an upper bound on the time that the transition may be enabled without being taken. A run of a timed transition system, then, is again a timed word. Alur et al. construct, for a given timed transition system T with a finite set of states, an event recording automaton that accepts precisely the runs of T. They present a translation from timed transition systems to event-recording automata, which leads to a PSPACE algorithm for checking if two timed transition systems have the same set of timed behaviors (they are equivalent).

Moreover, the authors provide an algorithm for checking if a timed automaton

meets a specification that is given as an event-clock automaton.

The class of event-clock automata, which contain both event-recording and event-predicting clocks, is a suitable specification language for real-time properties.

2.2 Temporal Logics

Properties of (runs of) transition systems are conveniently expressed in temporal logic. Temporal logics have proved to be useful for specifying concurrent systems, because they can describe the ordering of events in time without introducing time explicitly [Alur, 1991].

The use of temporal logic as a formalism for specifying the behavior of a reactive system over time was first proposed by Pnueli [1977].

Temporal logic is a modal logic, with modalities such as $\Diamond$, meaning "eventually", and $\Box$, meaning "always", providing a succinct and natural way of expressing the desired temporal requirements.

Two types of temporal logics have been proposed: *linear-time* and *branching-time*.

Linear-time logics are interpreted over linear structures of states. Every state sequence represents an execution sequence of a reactive system. A classical example of a linear-time logic is propositional temporal logic (PTL) [Gabbay et al., 1980].

In PTL, the typical response property that "every environment request p must be followed by a system response q" is defined by the formula

$$\Box(p \rightarrow \Diamond q),$$

which requires that in any possible behavior, if the system is in a state in which p is observed, then it will, at some later point, be in a state in which q is observed. Real-time extensions of PTL introduce a way of defining timing requirements such as the time-bounded response property that "every stimulus p is followed by a response q within 3 time units."

Formally, the real-time system S satisfies a linear-time formula φ iff every timed state sequence in T satisfies φ. Since the truth value of φ over a timed state sequence τ is completely determined by the observable component of τ, it suffices to consider the trace semantics of S and interpret φ over timed observation sequences: S satisfies φ if $\rho \models \varphi$ for every timed observation sequence ρ.

Branching-time temporal logics, on the other hand, are interpreted over tree structures of states. Every tree represents a reactive system, whose possible execution sequences correspond to the paths in the tree.

Since we require that each state $s \in S$ contains all the information necessary to decide the future behavior of the real-time system S, the set T of timed state sequences contains all the branching information for constructing a unique tree, with root s, whose paths represent the possible behaviors of S if started in state s. Thus, the truth value $s \models \varphi$ of a branching-time formula φ can be determined for each state $s \in S$. The real-time system S satisfies φ if $s \models \varphi$ for all $s \in S$.

It follows that linear-time logics employ an observation-oriented semantics, and branching-time logics employ a state-oriented semantics.

We will concentrate in this thesis on the linear-time framework.

2.2.1 Linear temporal logic (LTL)

Temporal logic is the class designation for modal logics whose modal operators are interpreted in a temporal manner: the basic operator $\Box$ is interpreted as "always" and, consequently, its dual $\Diamond$ means "eventually" [Alur and Henzinger, 1991]. The use of temporal logic as a formalism for specifying the behavior of reactive systems over time was first proposed by Pnueli [1977] and it has been studied extensively since then. Temporal logic provides a succinct and natural way of expressing the desired qualitative temporal requirements of speed-independent systems, including invariance, precedence, and responsiveness.

The traditional temporal operators, however, cannot refer to metric time and, hence, are insufficient for the specification of quantitative temporal requirements, or so-called hard real-time constraints, which put timing deadlines on the behavior of reactive systems.

Linear time temporal logic (LTL) extends classical logic by temporal modalities to refer to different (future or past) time points. Its formulae are interpreted over infinite sequences of states, such as the runs of transitions (or Kripke) systems.

Syntax. LTL formulas are built from the atomic propositions in AP, their negations, the temporal modalities $\bigcirc$ (next), $\mathcal{U}$ (until), and $\mathcal{R}$ (releases), and Boolean conjunction

and disjunction, defined by the following grammar:

$$\varphi \;:=\; p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi \,\mathcal{U}\, \varphi,$$

where $p \in AP$.

It is possible to express operators $\Box$, $\Diamond$ and $\mathcal{R}$ as follows:

- $\Diamond\varphi \equiv \mathit{true}\,\mathcal{U}\,\varphi$
- $\Box\,\varphi \equiv \neg(\mathit{true}\,\mathcal{U}\,\neg\varphi)$
- $\varphi\,\mathcal{R}\,\phi \equiv \neg(\neg\varphi\,\mathcal{U}\,\neg\phi)$

Semantics. LTL formulas are interpreted over timed traces and the semantics is defined with respect to an infinite sequence $\alpha \in (2^{AP})^{\omega}$ and a location $i \in \mathbb{N}$. For a formula φ, a timed trace (α, τ) the *satisfaction relation*, denoted $(\alpha, i) \models \varphi$ is used to indicate that the word α in the designated location i satisfies the formula φ.

- $(\alpha, i) \models p \;\Leftrightarrow\; p \in \alpha_i$;
- $(\alpha, i) \models \neg\varphi \;\Leftrightarrow$ not $(\alpha, i) \models \varphi$;
- $(\alpha, i) \models \varphi \wedge \psi \;\Leftrightarrow$ both $(\alpha, i) \models \varphi$ and $(\alpha, i) \models \psi$;
- $(\alpha, i) \models \varphi \vee \psi \;\Leftrightarrow$ either $(\alpha, i) \models \varphi$ or $(\alpha, i) \models \psi$;
- $(\alpha, i) \models \bigcirc\varphi \;\Leftrightarrow (\alpha, i+1) \models \varphi$;
- $(\alpha, i) \models \varphi\,\mathcal{U}\,\psi \;\Leftrightarrow$ there exists $k \geq i$ such that $(\alpha, k) \models \psi$, and for all $i \leq j < k$, we have $(\alpha, j) \models \varphi$;

A timed trace (α, τ) *satisfies* φ if $(\alpha, 0) \models \varphi$.

2.2.2 Properties expressed by LTL formulas

LTL formulas can express properties of paths in computation trees, hence they are relevant to temporal properties of transition systems. Since it is common to associate the set of all possible behaviors of a transition system with its computation tree, we

wonder what is the meaning of these formulas when we discuss properties of transition systems. A computation tree for a transition system $\mathcal{T}$ can be identified with the collection of its paths. It is known that these paths describe all possible computations of this transition system. Given an LTL formula φ, it is possible to consider at least two kinds of properties of $\mathcal{T}$:

1. does φ hold on some computation path for $\mathcal{T}$ from an initial state?

2. does φ hold on all computation paths for $\mathcal{T}$ from an initial state?

The two properties are dual to each other, indeed it is possible to argue that φ holds on some computation path if and only if it is not true that $\neg\varphi$ holds on all computation paths. Viceversa, φ holds on all computation paths if and only if it is not true that $\neg\varphi$ holds on some computation path. This means that, if we can express one of the properties, we can also express the other one. This is why it is possible to refer to temporal formulas as expressing properties of transition systems.

Reachability and safety properties. A state is called *reachable* if there is a computation path from an initial state leading to this state. Reachability is one of the most important properties of transition systems in connection with safety properties. Suppose that ϕ is a formula which expresses an undesirable property of a transition system. States satisfying ϕ are usually called unsafe or bad.

A *safety* property expresses that, under certain condition, an event never occurs. Then the system is *safe* if it is not possible to reach a state at which ϕ holds. Naturally, we would like to know whether the system is safe. It is possible to express the property of reachability of a state satisfying ϕ as the existence of a path satisfying $\Diamond\phi$. Then safety of the system can be expressed as non-reachability of a state satisfying ϕ, i.e., the property $\Box\neg\phi$. Naturally, this property must be held on all computation paths.

Another natural example of a safety property regards the use of a vending machine. Suppose to have a vending machine transition system and we would like to assure that the machine is never empty, i.e., there is always either coffee or beer in the storage. It can be expressed by the following formula:

$\Box(\phi_1 \vee \phi_2)$,

where ϕ_1 is the property of having coffee in the storage and ϕ_2 is the property of having beer in the storage.

Another example is the following: whenever there is a customer (represented by the formula θ), the machine has a drink. It can be expressed by the following LTL formula:

$\Box(\theta \Rightarrow \phi_1 \vee \phi_2)$.

Mutual exclusion. *Mutual exclusion* is usually formulated as a property of concurrent systems. It arises when two or more processes are not allowed to enter the same critical section of a concurrent system simultaneously. Assuming that there are two processes P_1, P_2, and that formulas φ_i, where $i = 1, 2$ denote that P_i is in the critical section, mutual exclusion can be expressed by the following LTL formula:

$\Box \neg(\varphi_1 \wedge \varphi_2)$.

An example of a natural mutual exclusion properties for the vending machine example is expressed by the following property: "coffee and beer cannot be in the dispenser simultaneously", whose corresponding formula is:

$\Box \neg(\chi_1 \wedge \chi_2)$,

where χ_1 is the property of having coffee in the dispenser and χ_2 is the property of having beer in the dispenser.

Deadlock. Generally, a concurrent program is in a *deadlock* situation when no terminal state is reached, yet no part of the program is able to proceed. A transition system is said to be deadlock-free if no computation in it leads to a deadlock. Deadlock-freeness is a special property, stating that the system can never be in a situation in which no progress is possible.

Assuming that the set of terminal states is represented by a temporal formula β and the property of having no state is represented by the formula ξ, it is possible to express deadlock-freedom by the LTL formula:

$\Box(\bigcirc \xi \Rightarrow \beta)$.

This formula must be true on every path. Indeed, it is easy to see that the formula $\bigcirc \xi$ represents the property that there is no next state, that is, no transition is possible. Likewise, it is possible to express reachability of a deadlock state as the existence of a state with the dual property:

$\Diamond(\bigcirc \xi \wedge \neg\beta)$.

Termination and finiteness. We know that there is not any notion of a terminal state, but it is possible to define terminal states as those from which no transition is possible. A terminal state can be represented by the formula $\bigcirc \xi$.

A transition system is called terminating, if every computation in it leads to a terminal state. *Termination* for a transition system is equivalent to the finiteness of all computation paths, which is equivalent to the *finiteness* of every computation tree from an initial state. But a computation path is finite if and only if it contains a deadlock state.

Therefore, to expresses the property that the computation tree is finite, we will use the the following formula, provided that this formula holds on every path:

$\Diamond \bigcirc \xi$.

Fairness. A *fairness* property expresses that, under certain conditions, an event will occur (or will fail to occur) infinitely often.

Usually, we are not interested in arbitrary computations of a transition system, as we know that some computations are impossible. Let us consider, for example, the case of the vending machine. We know that the vending machine must be recharged from time to time and this can be formulated as follows: on every computation path, the recharge transaction occurs infinitely many times. So the system must from time to time pass through a state which satisfies some property and this kind of constraints imposed on the system is called a *fairness* constraint, and computations satisfying fairness constraints are called fair. For the vending machine example, it is possible to impose many natural fairness constraints. For example, we may require that the dispenser contains a drink infinitely often, that students are customers infinitely often.

Fairness w.r.t. a property expressed by a formula ϕ means that ϕ holds infinitely often on all paths. We claim that fairness w.r.t. ϕ can be expressed by the following formula:

$\Box \Diamond \phi$.

Likewise, the property that expresses that ϕ holds infinitely often and the path is infinite is the following:

$\Box \bigcirc \Diamond \phi$.

Responsiveness. It is often the case in concurrent systems that one process sends requests that have to be acknowledged (or responded to) by other processes. For such systems we are interested in the *responsiveness* property: whether every request is eventually acknowledged. Assuming that the request is expressed by a formula Req and acknowledgement by a formula Ack, one can express responsiveness by the formula

$$\Box(Req \Rightarrow \bigcirc \Diamond Ack).$$

If we also want that request should remain true until it is acknowledged, responsiveness can be expressed by the formula

$$\Box(Req \Rightarrow (Req \,\mathcal{U}\, Ack)).$$

We can also require that the request formula and the acknowledgement formula be mutually exclusive, i.e., Req should remain true until it is acknowledged, after which it immediately becomes false. This can be expressed by the formula

$$\Box(Req \Rightarrow ((Req \wedge \neg Ack)\, \mathcal{U}(\neg Req \wedge Ack))).$$

2.2.3 Decision problems for LTL and known results

The automata-theoretic approach to linear temporal logic uses the theory of automata as a unifying paradigm for program specification, verification, and synthesis [Vardi, 1995]. Both programs and specifications are in essence descriptions of computations. These computations can be viewed as words over some alphabet and, thus, programs and specifications can be viewed as descriptions of languages over some alphabet. The automata-theoretic perspective considers the relationships between programs and their specifications as relationships between languages.

By translating programs and specifications to automata, questions about programs and their specifications can be reduced to questions about automata. More specifically, questions such as satisfiability of specifications and correctness of programs with respect to their specifications can be reduced to questions such as nonemptiness and containment of automata.

Unlike classical automata theory, which focused on automata on finite words, the applications to program specification, verification, and synthesis, use automata on infinite words, since the computations in which we are interested are typically infinite.

We are going to introduce decision problems for LTL formulas: satisfiability, veri-

fication and synthesis.

Formally, an LTL formula φ is *satisfiable* if there is some computation π such that $\pi \models \varphi$. A formula, that is unsatisfiable, usually is uninteresting as a specification, so unsatisfiability most likely indicates an erroneous specification. The satisfiability problem for LTL is to decide, given an LTL formula φ, whether φ is satisfiable.

Sistla and Clarke [1985] show that the satisfiability problem for LTL is PSPACE-complete.

Vardi and Wolper [1994] demonstrated that given an LTL formula φ , it is possible to construct a Büchi automaton A_φ, whose size is exponential in the length of φ, that accepts precisely the computations that satisfy φ. Thus, φ is satisfiable iff A_φ is not empty. This reduces the satisfiability problem to the nonemptiness problem. Since nonemptiness of Büchi automata can be tested in nondeterministic logarithmic space [Vardi and Wolper, 1994] and since A_φ is of exponential size, it is possible to have a polynomial-space algorithm.

To prove PSPACE-hardness, it can be shown that any PSPACE-hard problem can be reduced to the satisfiability problem. That is, there is a logarithmic-space algorithm that given a polynomial-space-bounded Turing machine M and a word w outputs an LTL formula $\varphi_{M,w}$ such that M accepts w iff $\varphi_{M,w}$ is satisfiable.

An LTL formula is *valid* if for every computation π we have that $\pi \models \varphi$. A valid formula is also uninteresting as a specification. The validity problem for LTL is to decide, given an LTL formula φ, whether φ is valid. It is easy to see that φ is valid iff $\neg\varphi$ is not satisfiable. Thus, the validity problem for LTL is also PSPACE-complete.

Moreover, given a finite-state program P and an LTL formula φ, the *verification* problem is to verify that all infinite words accepted by the automaton A_P satisfy the formula φ. It is possible to build a Büchi automaton A_φ that accepts exactly the computations satisfying the formula φ. The verification problem thus reduces to the automata-theoretic problem of checking that all computations accepted by the automaton A_P are also accepted by the automaton A_φ.

In some works [Lichtenstein and Pnueli, 1985; Sistla and Clarke, 1985; Vardi and Wolper, 1986], the authors have obtained the following results:

- The program complexity of the verification problem is complete for NLOGSPACE.
- The specification complexity of the verification problem is complete for PSPACE.

- Checking whether a finite-state program P satisfies an LTL formula φ can be done in time $O(|P| \cdot 2^{O(|\varphi|)})$ or in space $O((|\varphi| + \log |P|)^2)$.

In the verification problem it is given a finite-state program and an LTL specification and one have to verify that the program meets the specification.

A frequent criticism against this approach, however, is that verification is done after significant resources have already been invested in the development of the program. Since programs invariably contain errors, verification simply becomes part of the debugging process. The critics argue that the desired goal is to use the specification in the program development process in order to guarantee the design of correct programs. This is called *program synthesis*. It turns out that to solve the program-synthesis problem we need to use automata on infinite trees.

The classical approach to program synthesis is to extract a program from a proof that the specification is satisfiable. In [Pnueli and Rosner, 1989], [Abadi et al., 1989] and [Dill, 1989a], it is argued that the right way to approach synthesis of reactive programs is to consider the situation as an infinite game between the environment and the program, where the goal of the program is to satisfy the specification φ. It follows that we can assume without loss of generality that the winning condition for the game between the environment and the program is expressed by a Büchi automaton A: the program P wins the game if every run of P is accepted by A. We thus say that the program P *realizes* a Büchi automaton A if all its runs are accepted by A. We also say then that A is realizable. It turns out that the realizability problem for Büchi automata is essentially the solvability problem described in [Church, 1962].

Rabin [1972] showed that this problem can be solved by using Rabin tree automata.

We have thus reduced the realizability problem for LTL specifications to an automata theoretic problem: given a Büchi automaton A, decide if there is a tree $\mathcal{T}$ that realizes A. Then we reduce this problem to the nonemptiness problem for Rabin tree automata.

Abadi et al. [1989] and Pnueli and Rosner [1989] show that the realizability problem for Büchi automata can be solved in exponential time. Pnueli and Rosner [1989] show that the realizability problem for LTL can be solved in doubly exponential time and this time bound is essentially optimal.

We can resume the main results:

Theorem 2 *The satisfiability and the validity problems for* LTL *are* PSPACE*-complete.*

Theorem 3 *The* LTL *model checking problem is* PSPACE*-complete.*

Theorem 4 *The realizability problem for* LTL *can be solved in doubly exponential time.*

2.2.4 Metric temporal logic (MTL)

An important distinction among real-time models is whether one adopts a state-based semantics [Alur et al., 1996; Henzinger et al., 1998; Raskin and Schobbens, 1997] or an event-based semantics [Alur and Henzinger, 1993, 1994; Henzinger, 1991, 1998].

In the former, an execution of a system is modelled by a function that maps each point in time to the state propositions that are true at that moment. In the latter, one records only a countable sequence of events, corresponding to changes in the discrete state of the system.

The linear-temporal-logic approach to verification models an execution of a system by a sequence of states or events. This representation abstracts away from the precise times of observations, retaining only their relative order, so the LTL approach is inadequate to express specifications of systems whose correct behavior depends on quantitative timing requirements. There are several works where there is an adaptation of linear temporal logic to the real-time setting; see, e.g., [Alur and Henzinger, 1993, 1994; Alur et al., 1996; Koymans, 1990; Raskin and Schobbens, 1997; Wilke, 1994].

One of the earliest and most popular proposals for extending temporal logic to the real-time setting is to replace the temporal operators by time-constrained versions [Alur and Henzinger, 1991].

Koymans [1990] introduced the Metric Temporal Logic (MTL), as a prominent and successful instance of this approach. MTL extends LTL by constraining the temporal operators by (bounded or unbounded) intervals of the real numbers. An example is given by the following formula: $\Diamond_{[3,4]}\varphi$, which means that φ will hold within 3 to 4 time units from now.

Unfortunately, over the state-based semantics, the satisfiability and model-checking problems for MTL are undecidable [Henzinger, 1991].

This has led some researchers to consider various restrictions on MTL to recover decidability; see, e.g., [Alur et al., 1996; Henzinger et al., 1992; Wilke, 1994].

Undecidability arises from the fact that MTL formulas can capture the computations of a Turing machine: configurations of the machine can be encoded within a single unit-duration time interval, since the density of time can accommodate arbitrarily large amounts of information. Then, an MTL formula can specify that configurations be accurately propagated from one time interval to the next, in such a way that the timed words satisfying the formula correspond precisely to the halting computations of the Turing machine.

It turns out that the key ingredient required for this procedure to go through is *punctuality*: the ability to specify that a particular event is always followed exactly one time unit later by another one, expressed by the following formula

$\Box(p \Rightarrow \Diamond_{=1} q\varphi)$.

In the state-based and the event-based semantics, it has been claimed that, any logic strong enough to express the punctuality requirement will automatically be undecidable- see [Alur and Henzinger, 1991, 1993; Henzinger, 1998; Hirshfeld and Rabinovich, 2004] among others.

Decidability results for MTL involve placing restrictions on the semantics or the syntax of the logic to circumvent the problem of punctuality.

Alur and Henzinger [1993] showed that the satisfiability and model-checking problems for MTL relative to a discrete-time semantics are EXPSPACE-complete. Alur, Feder, and Henzinger introduced Metric Interval Temporal Logic (MITL) as a fragment of MTL in which the temporal operators may only be constrained by nonsingular intervals [Alur et al., 1996]. They showed that the satisfiability and model-checking problems for MITL relative to a dense-time semantics are also EXPSPACE-complete.

Wilke [1994] considered MTL over a dense-time semantics with bounded variability, i.e., the semantics is parameterized by a bound k on the number of events per unit time interval. He showed that the satisfiability problem is decidable in this semantics and that MTL with existential quantification over propositions is equally expressive as Alur-Dill timed automata.

However, while the claim regarding punctuality is correct over the state-based semantics, Ouaknine and Worrell [2007] show that it is erroneous in the event-based semantics. Indeed, they show that both satisfiability and model checking for MTL over

finite timed words are decidable, albeit with non-primitive recursive complexity. Over infinite words, they show that model checking the safety fragment of MTL, which includes invariance and punctual time-bounded response properties, is also decidable. MTL is also genuinely undecidable over the event-based semantics if in addition past temporal operators are allowed [Alur and Henzinger, 1993; Henzinger, 1991].

2.2.5 Metric interval temporal logic (MITL)

Alur et al. [1996] introduce the logic MITL, a linear temporal logic that is interpreted over timed state sequences. They use a linear or trace semantics for reactive systems, where the linear semantics of a system is a set of possible behaviors, each of which is represented by a sequence of system states. This model is most naturally extended to incorporate real time by associating, with every state, an interval of the real line, which indicates the period of time during which the system is in that state. They represent the possible behaviors of a real-time system by such timed state sequences, each of which defines a function from the nonnegative reals to the system states.

The problem, as we have seen also in the previous section, is that even the satisfiability of a very simple class of real-time properties turns out to be undecidable in this model [Alur and Henzinger, 1994].

An inspection of the undecidability proof shows that the only timing constraints required are of the form

$$\varphi = \Box(p \Rightarrow \Diamond_{=5} q),$$

predicting that every p-state is followed by a q-state precisely 5 time units later. In the first moment, this negative result has led to weaken the expressiveness of the model by adopting the semantic abstraction that, at every state change, we may record only a discrete approximation-the number of ticks of a digital clock to the real time. Thus, the formula φ is interpreted to require only that the p-state and the corresponding q-state are separated by exactly 5 clock ticks; their actual difference in time may be as much as 5.9 time units or as small as 4.1 time units. Under this weaker, digital-clock, interpretation, it has been shown that several interesting real-time logics are decidable [Alur and Henzinger, 1993, 1994].

Alur et al. [1996] pursue an alternative, syntactic, concession. Instead of digitizing the meaning of a sentence, they prohibit timing constraints that predict the time differ-

ence between two states with infinite accuracy. In particular, it is not possible to state the property given above, but only an approximation such as

$\varphi = \Box(p \Rightarrow \Diamond_{(4.9,5.1)} q)$

requiring that the p-state and the corresponding q-state are separated by more than 4.9 time units and less than 5.1 time units. They define a language that can constrain the time difference between events only with finite, yet arbitrary, precision. This is accomplished by prohibiting singular time intervals, of the form $[a, a]$ from constraining temporal speakers.

The resulting Metric Interval Temporal Logic (MITL) is shown to be decidable in EXPSPACE.

The complexity is PSPACEfor the fragment of MITL that employs only time intervals of the form $[a, \infty)$, (a, ∞), $[0, b)$, and $[0, b]$.

Properties of timed state sequences can, alternatively, be defined by timed automata [Alur and Dill, 1994].

We know that while the emptiness problem for timed automata is solvable, they are not closed under complement. MITL identifies a fragment of the properties definable by timed automata that is closed under all Boolean operations. The decision procedure for MITL leads to an algorithm for proving that a real-time system that is given as a timed automaton meets a requirements specification that is given in MITL. Thus, the novelty of their results is that they provide a logical formalism with a continuous interpretation of time that is suitable for the automatic verification and synthesis of real-time systems.

Syntax and Semantics A standard way of introducing real time into the syntax of temporal languages constrains the temporal operators with time intervals [Alur and Henzinger, 1993; Emerson et al., 1990; Koymans, 1990].

They adopt this approach for MITL, with the restriction that temporal operators cannot be constrained by singular intervals.

MITL formulas are built from the atomic propositions in AP, their negations, using Boolean connectives and a time-constrained version of the until operator $\mathcal{U}$ (until). The $\mathcal{U}$ operator may be constrained by any nonsingular interval with integer end-points. The restriction to integer end-points is used to simplify the presentation (the authors have showed, indeed, that their results extend to the case of rational end-points).

The formulas of MITL are defined by the following grammar:

$$\varphi \;:=\; p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \,\mathcal{U}_I\, \varphi,$$

where $p \in AP$ and I is a nonsingular interval with integer end-points (I may be unbounded).

The formulas of MITL are interpreted over timed state sequences, which provide an interpretation for the propositions at every time instant.

For a formula φ, a timed sequence α, and $t \in \mathbb{R}_+$, the *satisfaction relation*, denoted $(\alpha, t) \models \varphi$ is defined as follows:

- $(\alpha, t) \models p \;\Leftrightarrow\; p \in \alpha(t)$;
- $(\alpha, t) \models \neg\varphi \;\Leftrightarrow\; p \notin \alpha(t)$;
- $(\alpha, t) \models \varphi \wedge \psi \;\Leftrightarrow$ both $(\alpha, t) \models \varphi$ and $(\alpha, t) \models \psi$;
- $(\alpha, t) \models \varphi \,\mathcal{U}\, \psi \;\Leftrightarrow$ for some $t' \in I + t$, $(\alpha, v, t') \models \psi$, and $(\alpha, v, t'') \models \varphi$ for all $t < t'' < t'$;

A timed sequence α *satisfies* φ , denoted $\alpha \models \varphi$, if $(\alpha, 0) \models \varphi$. Note that MITL has no next-time operator, because the time domain is dense.

We write $L(\varphi)$ for the set of models of φ. The formula φ is satisfiable iff $L(\varphi) \neq \varnothing$; the two formulas φ and φ' are equivalent iff $L(\varphi) = L(\varphi')$.

The satisfiability problem for MITL is to decide whether or not a given MITL formula is satisfiable. The authors solve the satisfiability problem for MITL by reducing it to the emptiness problem for timed automata. Their main result is that, given an MITL formula ϕ, they construct a fair timed automaton B_ϕ, that accepts precisely the models of ϕ. The authors show that the satisfiability problem and the model checking problem for MITL are decidable and EXPSPACE-complete.

Examples of MITL formulas An interesting property that can be expressed by the MITL formalism is the following: the typical bounded-response requirement that "every p-state is followed by a q-state within 3 time units", expressed by the MITL-formula $\Box_{\geq 0}(p \Rightarrow \Diamond_{(0,3]} q)$.

Another requirement used to characterize events is expressed by the following MITL-formula which asserts that the proposition p is true in infinitely many singular states and nowhere else:

$\Diamond_{\geq 0} p \wedge \Box_{\geq 0}(p \Rightarrow (\neg p)\, \mathcal{U}\, p)$.

Moreover, it is possible to consider a time-out requirement. Supposing that p is a state constraint, and q is the time-out event, we wish to specify the requirement that "whenever p ceases to hold, either p becomes true again within less than 5 time units, or at time 5 the time-out event q happens". The following formula expresses the requirement:

$\Box_{\geq 0}((p \wedge \Box_{(0,5)}\, \neg p) \Rightarrow (\Box_{(0,5)}\, \neg q \wedge \Diamond_{(0,5]} q))$.

Additional examples of real-time requirements that are specifiable using time-constrained temporal operators can be found in [Koymans, 1990].

Chapter 3

Introduction of Parameters

In this chapter we consider the introduction of parameters in real-time formalisms. We take in account the use of parameters in timed models, such as parametric timed automata, and in temporal logics, such as parametric temporal logic. We also introduce the parametric notation used for our formalisms.

3.1 Motivations for the Use of Parameters

Considering a logic formalisms, the main motivation of the introduction of parameters, which are used to specify properties of a digital system and to model the system itself, is to allow a thorough analysis of the possible values of the numeric constants used in the design. The possibility of using such parametric constants is of great appeal mostly in the early stages of a design when, due to the scarce information on the system, the exact value of these constants is hard, or even impossible, to determine. In these regards, it is useful to be able to characterize the domains of the parameter valuations that make the considered analysis true.

The use of parametric constants has been advocated by many authors as a support to designers in the early stages of a design when not much is known on the system under construction (see for example [Alur et al., 1993c, 2001; Bruyère and Raskin, 2003; Courcoubetis and Yannakakis, 1992; Emerson and Trefler, 1999; Hune et al., 2002; Wang, 1996]). Unfortunately, the unrestricted use of parameters leads to undecidability [Alur et al., 1993c, 2001] and characterizing the domains of the parameter

valuations is possible only when we restrict to use only parameterized operators of just one polarity. The reason is essentially that if we can characterize such domains in an algorithmically usable format for the full logic, we would be able to answer the decision problems also when parameter valuations evaluating parameterized intervals to singular sets are allowed. This was first shown for the discrete-time logic parametric linear temporal logic (PLTL) [Alur et al., 2001], then for Büchi L/U automata [Bozzelli and La Torre, 2009], and we can replicate all the positive results shown there also for the logic parametric metric interval temporal logic (PMITL) [Di Giampaolo et al., 2010b] that will be introduced in Chapter 4.

Parameters are variables that abstract time values and can be used in Timed models and Temporal Logic formulas.

3.2 Use of Parameters in Timed Models

When the designer can not fully characterize the system in the early stages of the design or when the system may be affected by unknown features of the environment, it can be useful to introduce parameters in timed models. For the considered problems parameters are used in place of unknown or undefined values. The first parametric model is the *parametric timed automaton* (PTA), introduced by Alur et al. [1993c]; it allows parameters in clock constraints.

An interesting class of PTA is that of *lower bound/upper bound* (L/U) *automata* [Hune et al., 2002], which are defined as PTA such that the set L of the parameters which can occur as a lower bound in a parametric clock constraint is disjoint from set U of the parameters which can occur as an upper bound.

3.2.1 Parameters in timed automata

Traditional approaches to the verification of reactive and concurrent systems are limited to check qualitative properties such as safety and liveness, rather than timing properties, as is needed for the verification of real-time systems [Alur et al., 1993c]. This deficiency has been addressed over the last few years, and numerous formal approaches to the verification of real-time systems have been advocated (cf. [Abadi and Lamport, 1991; Alur, 1991; Henzinger, 1991; Hooman, 1991; Koymans, 1990; Ostroff, 1989;

Schneider et al., 1991]).

Most algorithms focus on the verification of concrete specifications, such as the property "an acknowledgement will be sent 10 milliseconds after a message has been received." Concrete timing constraints can be expressed and algorithmically verified using real-time temporal logics [Alur and Henzinger, 1993, 1994; Alur et al., 1996; Emerson et al., 1990; Henzinger et al., 1994b; Wang et al., 1993] or time-constrained finite-state machines [Alur and Dill, 1990; Alur and Henzinger, 1992; Alur et al., 1990; Dill, 1989b; Lewis, 1990].

Real-time systems, however, are typically embedded in larger environments, and the system designer has to design the system relative to certain parameters of the environment. Thus arises the real need for verifying parametric specifications. As an example, consider a real-time system S, and suppose to verify a property p of the system as long as the deadline d of an action is less than the delay r in receiving an acknowledgement, $r > d$ [Jahanian, 1989].

The design of a robust system requires the verification of the desired behavior of the system without concrete values for the parameters r and d. Indeed, when studying the literature on real-time protocols, one sees that the desired timing properties for protocols are almost invariably parametric (cf. [Attiya et al., 1991; Strong et al., 1990; Weinberg and Zuck, 1992]) , because concrete timing constraints make sense only in the context of a given concrete environment.

Unfortunately, the parametric verification problem is very difficult. In fact, it is easy to show that standard real-time temporal logics become undecidable even when a single parameter is introduced. Hence, rather than temporal logic, as a model of this theory are introduced PTA. Parametric timed automata generalize the timed automata of Alur and Dill [1990], which have emerged as an attractive model for real-time systems. Timed automata are finite-state machines that are equipped with clocks, which are used to constrain the accepting runs by imposing timing requirements on the transitions. While the timing requirements of timed automata are concrete (for example it is possible to enable a transition for 10 time units) , the timing requirements of parametric timed automata are parametric (for example it is possible to enable a transition for d time units, for some parameter d). A parametric timed automaton characterizes a set of parameter values, namely, those for which the automaton has an accepting run. Thus, a parametric timed automaton is, like a system of equations or inequalities, simply a

constraint on admissible parameter values.

The problem of emptiness for parametric timed automata is defined as follows: given a parametric timed automaton, are there concrete values for the parameters so that the automaton has an accepting run? The solution of this problem allows the verification of parametric real-time specifications. The emptiness problem reveals that the number of clocks in a parametric timed automaton is critical to the decidability of the problem.

In the work of Alur et al. [1993c] it is demonstrated that for automata with one parametrically constrained clock (and possibly many concretely constrained clocks), emptiness is decidable whereas three parametrically constrained clocks are sufficient to bring about undecidability. The authors describe a symbolic fixpoint computation procedure to solve the emptiness problem. The procedure is sound, and though its termination is not guaranteed in general, it terminates for many examples of practical interest. They provide parametric verifications of a railroad gate controller and of Fischers timing-based mutual-exclusion protocol. The decidability in the case of two clocks is open, and it reveals intriguing connections with hard decision problems in logic (existential Presburger arithmetic with divisibility) and automata theory (special classes of nondeterministic two-way l-counter machines).

Since clocks are used to measure delays between events, the number of clocks is a fair indicator of the structural complexity of the timing constraints imposed on a system.

3.2.2 Lower bound/upper bound parametric timed automata (L/U PTA)

Hune et al. [2002] identify a subclass of parametric timed automata, called lower bound/upper bound (L/U) automata, in which each parameter occurs either as a lower bound or as an upper bound in the timing constraints. Despite this limitation, the model is still interesting in practice. In fact, L/U automata can be used to model the Fisher's mutual exclusion algorithm [Lamport, 1987], the root contention protocol [Society, 1996], and other known examples from the literature (see [Hune et al., 2002]).

Hune et al. show that the emptiness problem for L/U automata with respect to finite runs is decidable. The case of infinite accepting runs (which is crucial for the

verification of liveness properties) is not investigated, and does not follow from their results.

Bozzelli and La Torre [2009] further investigate the class of L/U automata and consider acceptance conditions over infinite runs. Given an L/U automaton A, denoting with $\Gamma(A)$ the set of parameter valuations for which the automaton has an infinite accepting run, they show that questions about $\Gamma(A)$ can be answered considering a bounded set of parameter valuations of size exponential in the size of the constants and the number of clocks, and polynomial in the number of parameters and locations of A.

Therefore, the authors are able to show that checking the set $\Gamma(A)$ for emptiness, universality (i.e., if $\Gamma(A)$ contains all the parameter valuations), and finiteness is PSPACE-complete. The main argument for such results is as follows: suppose that A is an L/U automaton which uses parameters only as lower bounds (resp., upper bounds); then if an infinite run ρ is accepted by A for large-enough values of the parameters, it is possible to determine appropriate finite portions of ρ which can be "repeatedly simulated" (resp., "deleted") thus obtaining a run ρ' which is accepted by A for larger (resp., smaller) parameters values. Parameters in system models can be naturally related by linear equations and inequalities.

Moreover, Bozzelli and La Torre [2009] consider consider *constrained emptiness* and *constrained universality* on L/U automata, where the constraint is represented by a linear system over parameters. They show that these problems are in general undecidable, and become decidable in polynomial space (and thus PSPACE-complete) if they do not compare parameters of different types in the linear constraints. In addition, they show that when all the parameters in the model are of the same type (i.e., either lower bound or upper bound), it is possible to compute an explicit representation of the set $\Gamma(A)$ by linear systems over parameters whose size is doubly exponential in the number of parameters.

3.3 Use of Parameters in Temporal Logic Formulas

Uncertainties in the system requirements due to changing demands or changing environments causes the need for verifying parametric specification.

The problem is that unrestricted use of parameters leads to undecidability.

Alur et al. [2001] extend the standard model checking paradigm of linear temporal

logic, LTL, to a "model measuring" paradigm where one can obtain more quantitative information beyond a "Yes/No" answer.

They extend linear temporal logic to parametric temporal logic (PLTL), in which temporal operators can be subscripted, together with a direction, by a variable ranging over the natural numbers. The algorithms they propose exhibit the same PSPACE complexity as LTL model checking.

Kupferman et al. [2009] introduce and study an extension of LTL, called PROMPT-LTL. In addition to the usual temporal operators of LTL, the logic PROMPT-LTL has a new temporal operator that is used for specifying eventualities with a bounded wait time. They term the operator prompt eventually and denote it by F_p. The authors study various problems related to PROMPT-LTL, including realizability, model checking, and assume-guarantee model checking, and show that they can be solved by techniques that are quite close to the standard techniques for LTL.

A parametric extension of the logic $\text{MITL}_{0,\infty}$ is introduced by Bozzelli and La Torre [2009], which prove that the related satisfiability and model checking (w.r.t. L/U automata) problems are PSPACE-complete.

3.3.1 Parametric linear temporal logic (PLTL)

Traditional temporal logics, such as linear temporal logic (LTL), allow only qualitative assertions about the temporal ordering of events.

For example, is is possible to express in LTL a typical temporal requirement as the following one, "every request p is followed by a response q", by the formula

$$\Box(p \rightarrow \Diamond q)$$

However, that formula does not specify any bound on how soon the response should follow the request. In various circumstances, for assessing the efficiency and practicality of the design being modeled, it may be useful to have additional quantitative guarantees. Consequently, a variety of real-time or quantitative temporal logics have been proposed (see, for instance, [Alur and Henzinger, 1993; Alur et al., 1996; Emerson, 1990; Koymans, 1990]).

A representative of such logics is metric temporal logic (MTL) [Alur and Henzinger, 1993; Koymans, 1990].

Using MTL it is possible to limit the scope of temporal operators by subscripting

them with natural numbers and a direction (before ($\leq$) or after ($\geq$)). For that reason, we can express the quantitative requirement that "every p is followed by a response q, within 5 steps" by the MTL formula

$\Box(p \rightarrow \Diamond_{\leq 5}\, q)$.

However, even with quantitative temporal logics, model checking still yields only a "yes/no" answer. It may happen that we have no idea whether 5 is the best bound on the response time, so we would actually like to find out what quantitative bounds can be placed on the eventuality.

Alur et al. [2001] have presented the first decidable framework for incorporating parametric reasoning in linear temporal logic, extending the "yes/no" paradigm of model checking to a "model measuring" paradigm. They extend linear temporal logic to parametric temporal logic (PLTL), in which temporal operators can be subscripted, together with a direction, by a variable ranging over the natural numbers. In particular, $\Diamond_{\leq x}$ will mean "in at most x steps p occurs", and $\Box_{\leq y}$ will mean "for at least y steps q holds". In this case the variables serve to delimit the scope of the temporal operators, but make no a priori claim as to what their values should be. The response property we have seen before is thus written in PLTL as

$\Box(p \rightarrow \Diamond_{\leq x}\, q)$

The question then becomes: "for what values of x does the formula hold for the system being modeled?". For an operator such as $\Diamond_{\leq x}$, we would like to compute the minimum satisfying value of x, while for an operator such as $\Box_{\leq y}$, we would like to compute the maximum satisfying value of y.

In general, a system model K and a formula $\varphi(x_1, \ldots, x_k)$ with multiple parameters define the set $V(K, \varphi)$ of valuations α of $(x_1, \ldots, x_k)$ under which φ holds for K. Alur et al. [2001] present algorithms to answer the following questions:

- Is there a parameter valuation α for which K satisfies φ? (i.e., is $V(K, \varphi)$ nonempty?)
- Does K satisfy φ for any α? (i.e., does $V(K, \varphi)$ contain all valuations?)
- Is the set $V(K, \varphi)$ finite (or bounded)?
- Find a parameter valuation $\alpha \in V(K, \varphi)$ which minimizes the maximum (or maximizes the minimum) parameter value.

Given a parametric formula φ and a model K, the authors show how to find whether parameter valuations exist under which the formula φ holds for the model K, and if so to find ones which satisfy various optimality criteria. The complexity of their algorithms for the above problems is essentially that of ordinary LTL model checking: PSPACE in the formula size and polynomial-time in the size of the model. Their optimization algorithms, such as finding max-min or min-max valuations, are polynomial in the size of the model and exponential in the size of the formula. When all parameterized operators in the formula are of the same polarity, they show how to compute an explicit representation of the set $V(K, \varphi)$ by symbolic constraints on parameter values; this representation is polynomial in the size of the model, exponential in the size of the formula, and doubly exponential in the number of parameters. The concept of polarity is semantic and is related to whether the space of the values for a parameter such that the formula is satisfied is upward or downward closed. The key to their upper bounds is a form of pumping lemma: if a sequence satisfies a formula for large enough values of the parameters, it is possible to repeat (or delete) appropriate cycles to obtain a sequence that satisfies the formula for larger (or smaller) parameter values.

The logic PLTL is defined with two restrictions and removing any of these restrictions the resulting logic becomes undecidable. The fist is that equality subscripts (e.g., $\Diamond_{=x}$) are not allowed while the second one is that the same parameter cannot appear in association with two operators with different polarities (e.g., both $\Diamond_{\leq x}$ and $\Box_{\leq y}$).

3.3.2 Prompt linear temporal logic (PROMPT-LTL)

Temporal logic, in its many different flavors, has been widely accepted as an appropriate formal framework for the description of on-going behavior of reactive systems [Manna and Pnueli, 1992a], since its introduction into computer science [Pnueli, 1977].

Temporal properties are traditionally classified into safety and liveness properties. Intuitively, safety properties assert that nothing bad will ever happen during the execution of the system, and liveness properties assert that something good will happen eventually. Temporal properties are interpreted with respect to systems that generate infinite computations. In satisfying liveness properties, there is no bound on the ”wait time”, namely the time that may elapse until an eventuality is fulfilled.

Consider the LTL formula $\Diamond a$. It asserts that a holds eventually, but it does not

specify any bound on the time when a will hold.

This has given rise to formalisms in which the eventually operator $\Diamond$ is replaced by a bounded-eventually operator $\Diamond_{\leq k}$. The operator is parameterized by some $k \geq 0$, and it bounds the wait time to k [Beer et al., 1994; Emerson et al., 1990].

Since we assume that time is discrete, the operator $\Diamond_{\leq k}$ is simply a syntactic sugar for an expression in which the next operator X is nested. A drawback of the above formalism is that the bound k needs to be known in advance, which is not the case in many applications. Indeed, it may depend on the system, which may not yet be known, or it may change, if the system changes. In addition, the bound may be very large, causing the state-based description of the specification (e.g., an automaton for it) to be very large too. Thus, the common practice is to use liveness properties as an abstraction of such safety properties: one writes $\Diamond a$ instead of $\Diamond_{\leq k} a$, for an unknown or a too large k. It follows that the abstraction of safety properties by liveness properties is not sound in the linear-time approach.

This is troubling for designers because it is a common practice for them to interpret an eventuality $\Diamond a$, as an abstraction of a bounded eventuality $\Diamond_{\leq k} a$, for an unknown k, and satisfaction of a liveness property is often not acceptable unless we can bound its wait time.

Kupferman et al. [2009] introduce and study an extension of LTL that addresses the above problem, called prompt linear temporal logic (PROMPT-LTL). In addition to the usual temporal operators of LTL, their logic, PROMPT-LTL, has a new temporal operator that is used for specifying eventualities with a bounded wait time. They term the operator *prompt eventually* and denote it by F_p.

Formally, the semantics of PROMPT-LTL is defined as follows. For a PROMPT-LTL formula ϕ and a bound $k \geq 0$, let ϕ^k be the LTL formula obtained from ϕ by replacing all occurrences of F_p by $F^{\leq k}$. Then, a system S satisfies ϕ iff there is $k \geq 0$ such that S satisfies ϕ^k . A system S satisfies a PROMPT-LTL formula ϕ if there is some bound k on the wait time for all prompt-eventually subformulas of ϕ in all computations of S.

It is important to note that while the syntax of PROMPT-LTL is very similar to that of LTL, its semantics is defined with respect to an entire system, and not with respect to computations. Indeed, promptness plays no role in the context of a single computation: if the computation satisfies an eventuality, it ought to satisfy it with some bounded wait

time, namely the time that has elapsed until the eventuality has been satisfied.

Unlike linear temporal logics, it is not possible to characterize a PROMPT-LTL formula ϕ over a set AP of atomic propositions by a set of computations $L_\phi \subseteq (2^{AP})^\omega$ such that a system S satisfies ϕ iff the language of S is contained in L_ϕ . On the other hand, unlike branching temporal logics, if two systems agree on their languages, then they agree also on the satisfaction of all PROMPT-LTL formulas. Thus, PROMPT-LTL intermediates between the linear and branching approaches: as in the linear approach, the specification refers to the set of computations of the system rather than its computation tree; as in the branching approach, we cannot consider these computations individually, indeed to conclude that a PROMPT-LTL formula holds over a set of computations we cannot evaluate it over each computation separately.

The authors study various problems related to PROMPT-LTL, including realizability, model checking, and assume-guarantee model checking, and show that they can be solved by techniques that are quite close to the standard techniques for LTL. Satisfiability of PROMPT-LTL is easily reduced to satisfiability of LTL. Indeed, consider a PROMPT-LTL formula ϕ and the LTL formula ϕ' obtained from ϕ by replacing all occurrences of F_p by F. It is well known that if ϕ' is satisfiable, it is satisfiable over a single regular computation (i.e., a prefix and a suffix that repeats infinitely often), cf. [Vardi and Wolper, 1994]. It is easy to see that the same computation satisfies ϕ .

For the other problems, the authors develop a new technique, described as follows. Consider a PROMPT-LTL formula ϕ over AP. Let p be an atomic proposition not in AP. Think about p as a description of one of two colors, say green (p holds) and red (p does not hold). Each computation of the system can be partitioned to blocks such that states of the same block agree on their color. They show that a system S satisfies a PROMPT-LTL formula ϕ iff there is some bound $k \geq 0$ such that it is possible to color each computation π of S so that the induced blocks are of length k, and whenever a suffix of π has to satisfy an eventuality, the eventuality is fulfilled within two blocks. Indeed, the latter condition holds iff all eventualities have wait time at most $2k$.

The key idea behind their technique is that rather than searching for a bound k for the prompt eventualities, which can be quite large, it is enough to make sure that there is a coloring in which all blocks are of a (not necessarily bounded) finite length, and then use some regularity argument in order to conclude that the size of the blocks could actually be bounded. Forcing the blocks to be of a finite length can be done

by requiring the colors to alternate infinitely often. As for regularity, in the case of realizability, regularity follows from the finite-model property of tree automata. In the case of model checking and assume-guarantee model checking, regularity follows from the finiteness of the system.

The complexity results that follow from their algorithms show that reasoning about PROMPT-LTL is not harder than reasoning about LTL: realizability is 2EXPTIME-complete, and model checking and assume-guarantee model checking are PSPACE-complete. For LTL, many heuristics have been studied and applied. Some of them are immediately applicable for PROMPT-LTL (c.f., optimal translations of formulas to automata), and some should be extended to the prompt setting (e.g., bad-cycle detection algorithms).

They also study some theoretical aspects of PROMPT-LTL, such as a bound on the wait time, when exists (may be linear in the system and exponential in the PROMPT-LTL formula), the ability to translate PROMPT-LTL formulas to branching-temporal logics (a translation to the μ-calculus is always possible, but may involve a significant blow up), and the ability to determine whether a PROMPT-LTL formula has an equivalent LTL formula (PSPACE-complete).

3.3.3 A parametric extension of a fragment of the metric interval logic ($\mathbf{P_{0,\infty}MITL_{0,\infty}}$)

The logic $P_{0,\infty}MITL_{0,\infty}$is defined by Bozzelli and La Torre [2009] as a parametric extension of the logic $MITL_{0,\infty}$.

We will define the logic $P_{0,\infty}MITL_{0,\infty}$ and the known results in Section 5.1.

3.4 Parametric notations

In this section, we introduce some concepts regarding the notations of parametric formalisms.

3.4.1 Parameterized Intervals

We consider non-empty intervals (convex sets) of non-negative real numbers. We use the standard notation $[a, b]$, $]a, b[$, $[a, b[$, and $]a, b]$ to denote respectively the closed, open, left-closed/right-open and left-open/right-closed intervals with end-points a and b. When we do not need to specify if an end-point is included or not in an interval, we simply use parentheses: for example, we denote with (a, b) any of the possible intervals with end-points a and b. A *time interval* I is an interval (a, b) such that $0 \leq a \leq b$, and $a < b$ if I is not closed. A closed time interval $I = [a, a]$ is called *singular*. Given an interval $I = (a, b)$ and $t \geq -a$, with $I + t$ we denote the interval $(a + t, b + t)$ such that I is left-closed (resp. right-closed) iff $I + t$ is left-closed (resp. right-closed).

An *interval sequence* is an infinite sequence $I_0, I_1 \ldots$ of time intervals such that:

- for all i, $I_i \cap I_{i+1} = \varnothing$ and, denoting $I_i = (a_i, b_i)$, $a_{i+1} = b_i$ holds (*along the time line I_{i+1} follows I_i*);
- each real number $t \geq 0$ belongs to some interval I_i (*the sequence of intervals covers the reals*).

We fix a set of atomic propositions AP. A *timed sequence* over AP is an infinite sequence $\alpha = (\alpha_0, I_0)(\alpha_1, I_1) \ldots$ such that $\alpha_i \in 2^{AP}$, for all i, and $I_0, I_1 \ldots$ is an interval sequence. For each $t \geq 0$, $\alpha(t)$ denotes the unique α_i such that $t \in I_i$.

3.4.2 Parametric Expression

We use the symbols U and L to denote two disjoint sets of parameters. A *parametric expression* is an expression of the form $c + x$, where $c \in \mathbb{N}$ and x is a parameter. With $\mathcal{E}(U)$ (resp. $\mathcal{E}(L)$) we denote the set of all the parametric expressions over parameters from U (resp. L). A *parameterized interval* is a time interval (a, b) such that either a or b belong to $\mathcal{E}(L) \cup \mathcal{E}(U)$. In the following, we sometimes use the term interval to indicate either a parameterized interval or a time interval. A *parameter valuation* $v : L \cup U \longrightarrow \mathbb{N}$ assigns a natural number to each parameter. Given a parameter valuation v and an interval I, with I_v we denote the time interval obtained by evaluating the end-points of I by v (in particular, if I is a time interval then $I_v = I$).

In the following, for parameter valuations v, v' and $\approx \in \{<, \leq, =, \geq, >\}$, we use the standard notation $v \approx v'$ to denote the extension of $\approx$ to tuples, that is, the relation defined as: $v(x) \approx v'(x)$ for each parameter x.

3.4.3 Parametric Timed Automata

Parametric Timed Automata extend Timed Automata, allowing the use of parameters in the clock constraints, see [Alur et al., 1993c]. We briefly recall the definition. Given a finite set of clocks X and a finite set of parameters P, a *clock constraint* is a positive boolean combination of terms of the form $\xi \approx e$ where $\xi \in X$, $\approx \in \{<, \leq, \geq, >\}$, and either $e \in \mathbb{N}$ or e is a parametric expression. In the following, with Ξ we denote the set of all clock constraints over X and P.

A *parametric timed automaton* (PTA) is a tuple $\mathcal{A} = \langle Q, Q^0, X, P, \beta, \Delta, \lambda \rangle$, where Q is a finite set of *locations*, $Q^0 \subseteq Q$ is the set of *initial locations*, X is a finite set of clocks, P is a finite set of parameters, β is a function assigning to each location q a *parametric clock constraint* $\beta(q)$, $\Delta \subseteq Q \times \Xi \times Q \times 2^X$ is a *transition relation*, and $\lambda : Q \to 2^{AP}$ is a function labeling each location with a set of atomic propositions.

A *clock interpretation* $\sigma : X \to \mathbb{R}_+$ assigns real values to each clock. A clock interpretation $\sigma + t$, for $t \in \mathbb{R}_+$, assigns $\sigma(\xi) + t$ to each $\xi \in X$. For $\gamma \subseteq X$, $\sigma[\gamma := 0]$ denotes the clock interpretation that assigns 0 to all clocks in γ, and $\sigma(\xi)$ to all the other clocks ξ. Given $q \in Q$, we say that a clock interpretation σ and a parameter valuation v satisfy a parametric clock constraint δ, denoted $(\sigma, v) \models \delta$, iff evaluating each clock of δ according to σ and each parameter of δ according to v, the resulting boolean expression holds true.

For locations $q_i \in Q$, clock interpretations σ_i, clock constraints $\delta_i \in \Xi$, clock sets $\gamma_i \subseteq X$, and intervals I_i, a *run* ρ of a PTA $\mathcal{A}$, under a parameter valuation v, is an infinite sequence $\xrightarrow{\sigma_0} (q_0, I_0) \xrightarrow[\delta_1, \gamma_1]{\sigma_1} (q_1, I_1) \xrightarrow[\delta_2, \gamma_2]{\sigma_2} (q_2, I_2) \xrightarrow[\delta_3, \gamma_3]{\sigma_3} \ldots$, such that:

- $q_0 \in Q^0$, and $\sigma_0(\xi) = 0$, for each clock ξ;
- $I_0, I_1 \ldots$ is an interval sequence;
- for all $i \geq 0$, denoting $I_i = (a_i, b_i)$ and $\sigma'_i = \sigma_i + b_i - a_i$:

  (1) $(q_i, \delta_{i+1}, q_{i+1}, \gamma_{i+1}) \in \Delta$, $(\sigma'_i, v) \models \delta_{i+1}$ and $\sigma_{i+1} = \sigma'_i[\gamma_{i+1} := 0]$;

(2) $\forall t \in I_i$, $\sigma_i + (t - a_i)$ and v satisfy $\beta(q_i)$.

The timed sequence associated with ρ is $(\lambda(q_0), I_0)(\lambda(q_1), I_1)(\lambda(q_2), I_2) \ldots$.

Recall that two timed sequences α' and α'' are equivalent if $\alpha'(t) = \alpha''(t)$ for all $t \geq 0$. With $L(\mathcal{A}, v)$ we denote the set of of all timed sequences over AP which are equivalent to those associated with a run of $\mathcal{A}$ under a parameter valuation v.

An interesting class of PTA is that of *lower bound/upper bound* (L/U) *automata* which are defined as PTA such that the set L of the parameters which can occur as a lower bound in a parametric clock constraint is disjoint from set U of the parameters which can occur as an upper bound. For L/U automata both acceptance criteria on finite runs and on infinite runs have been considered [Bozzelli and La Torre, 2009; Hune et al., 2002]. Here, we recall the Büchi acceptance condition. A *Büchi* L/U automaton $\mathcal{A}$ is an L/U automaton coupled with a subset F of locations. A run ρ is accepting for $\mathcal{A}$ if at least one location in F repeats infinitely often along ρ. We denote with $\Gamma(\mathcal{A})$ the set of parameter valuations v such that there exists an accepting run under v. We recall the following result.

Theorem 5 ([Bozzelli and La Torre, 2009]) *The problems of checking the emptiness and the universality of* $\Gamma(\mathcal{A})$*, for a Büchi* L/U *automaton* $\mathcal{A}$*, are* PSPACE*-complete.*

Chapter 4

Parametric Dense-Time Metric Interval Temporal Logic

In this Chapter we introduce the logic PMITL as a parametric extension of MITL. We allow the intervals in the subscripts of the temporal operators to have as an endpoint a *parametric expression* of the form $c + x$, for a parameter x and a constant c.

For this logic, we study decision problems which are the analogous of satisfiability, validity and model-checking problems for non-parametric temporal logic and we show the concept of polarity of parameterized temporal operators. Moreover, we give an example regarding the use of PMITL and we show interesting properties that can be expressed using our formalism. We prove the decidability results of the satisfiability, validity, and model-checking problems, using a translation to the emptiness and the universality problems for Büchi L/U automata.

4.1 Syntax of PMITL

The *Parametric dense-time Metric Interval Temporal Logic* (PMITL)[Di Giampaolo et al., 2010b] extends MITL [Alur et al., 1996] by allowing parameterized time intervals as subscripts of temporal operators. The PMITL formulas over AP are defined by the following grammar:

$$\varphi \;:=\; p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \,\mathcal{U}_I\, \varphi \mid \varphi \,\mathcal{R}_J\, \varphi,$$

where $p \in AP$, and H and J are either non-singular time intervals with end-points in $\mathbb{N} \cup \{\infty\}$ or parameterized time intervals such that:

- for $H = (a, b)$ either (1) $a \in \mathbb{N}$ and $b \in \mathcal{E}(U)$ or (2) $a \in \mathcal{E}(L)$ and $b \in (\mathbb{N} \cup \{\infty\})$;
- for $J = (a, b)$ either (1) $a \in \mathbb{N}$ and $b \in \mathcal{E}(L)$ or (2) $a \in \mathcal{E}(U)$ and $b \in (\mathbb{N} \cup \{\infty\})$.

As usual, we use the abbreviations $\Diamond_H \varphi$ and $\Box_J \varphi$ for $true\, \mathcal{U}_H\, \varphi$ and $false\ \mathcal{R}_J\ \varphi$, respectively.

For a given formula φ, a parameter valuation v is *admissible* for φ if for each interval I in the subscripts of φ, I_v is either a non-singular time interval or the interval $[0, 0]$. With $\mathcal{D}(\varphi)$ we denote the set of all the admissible valuations for φ.

The logic MITL$_{0,\infty}$ is defined by Alur et al. [1996] as a syntactic restriction of MITL where all the time intervals (a, b), that are used as subscripts of the temporal operators, are such that either $a = 0$ or $b = \infty$. With PMITL$_{0,\infty}$ we denote the parametric extension of MITL$_{0,\infty}$ which corresponds to the fragment of PMITL where all the time (non-parameterized) intervals are as in MITL$_{0,\infty}$. We remind that we use the acronym P$_{0,\infty}$MITL$_{0,\infty}$ to denote the parametric extension of MITL$_{0,\infty}$ where also the parameterized intervals are restricted such that one of the end-points is either 0 or ∞, that is introduced in Section 3.3.3.

We consider also two interesting fragments of PMITL$_{0,\infty}$: PMITL$_{\Diamond}$ is the fragment where the only parameterized operators are either of the form $\Diamond_{(c,d+x)}$, for $x \in U$, or one of the interval end-points is 0 or ∞, and PMITL$_{\Box}$ is the fragment of PMITL$_{0,\infty}$ where the only parameterized operators are either of the form $\Box_{(c,d+y)}$, for $y \in L$, or one of the interval end-points is 0 or ∞.

All the considered fragments of PMITL are reported in Figure 4.1, where the edges denote syntactic inclusion between the connected classes, from the top to the bottom.

In the following, given a formula φ we denote with K_φ the maximal constant used in φ augmented by 1 and with N_φ the number of subformulas of φ.

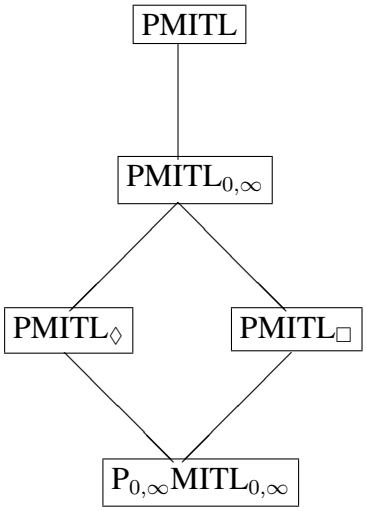

Figure 4.1: *The hierarchy of PMITL fragments with respect to syntactic inclusion.*

4.2 Semantics of PMITL

PMITL formulas are interpreted over timed sequences and with respect to a parameter valuation. For a formula φ, a timed sequence α, a parameter valuation v, and $t \in \mathbb{R}_+$, the *satisfaction relation under valuation* v, denoted $(\alpha, v, t) \models \varphi$, is defined as follows:

- $(\alpha, v, t) \models p \;\Leftrightarrow\; p \in \alpha(t)$;
- $(\alpha, v, t) \models \neg p \;\Leftrightarrow\; p \notin \alpha(t)$;
- $(\alpha, v, t) \models \varphi \wedge \psi \;\Leftrightarrow$ both $(\alpha, v, t) \models \varphi$ and $(\alpha, v, t) \models \psi$;
- $(\alpha, v, t) \models \varphi \vee \psi \;\Leftrightarrow$ either $(\alpha, v, t) \models \varphi$ or $(\alpha, v, t) \models \psi$;
- $(\alpha, v, t) \models \varphi \,\mathcal{U}_{\boldsymbol{H}}\, \psi \;\Leftrightarrow$ for some $t' \in \boldsymbol{H}_v + t$, $(\alpha, v, t') \models \psi$, and $(\alpha, v, t'') \models \varphi$ for all $t < t'' < t'$;
- $(\alpha, v, t) \models \varphi \,\mathcal{R}_{\boldsymbol{J}}\, \psi \;\Leftrightarrow$ either $(\alpha, v, t') \models \psi$, for all $t' \in \boldsymbol{J}_v + t$, or there exists $t'' > t$ such that $(\alpha, v, t'') \models \varphi$ and $(\alpha, v, t') \models \psi$ for all $t' \in [t, t''] \cap \boldsymbol{J}_v + t$.

A timed sequence α *satisfies* φ *under valuation* v, denoted $(\alpha, v) \models \varphi$, if $(\alpha, v, 0) \models \varphi$. Observe that the temporal operators $\mathcal{U}$ and $\mathcal{R}$ are dual.

Two PMITL formulas φ_1 e φ_2 are *equivalent* if $\mathcal{D}(\varphi_1) = \mathcal{D}(\varphi_2)$ and for all the timed sequences α and all the parameter valuations $v \in \mathcal{D}(\varphi_1)$, $(\alpha, v) \models \varphi_1$ if and only if $(\alpha, v) \models \varphi_2$.

4.3 Decision Problems

For the logic PMITL, we study the satisfiability, the validity and the model-checking problems with respect to an L/U automaton. More precisely, given a PMITL formula φ and an L/U automaton $\mathcal{A}$, consider the following sets:

- the set $S(\varphi)$ of the parameter valuations $v \in \mathcal{D}(\varphi)$ such that there is a timed sequence that satisfies φ under valuation v;
- the set $V(\varphi)$ of the parameter valuations $v \in \mathcal{D}(\varphi)$ such that all the timed sequences satisfy φ under valuation v;
- the set $S(\mathcal{A}, \varphi)$ of the parameter valuations $v \in \mathcal{D}(\varphi)$ such that there is a timed sequence in $L(\mathcal{A}, v)$ that satisfies φ under the valuation v;
- the set $V(\mathcal{A}, \varphi)$ of the parameter valuations $v \in \mathcal{D}(\varphi)$ such that all the timed sequences in $L(\mathcal{A}, v)$ satisfy φ under the valuation v.

For each of these sets, we study the *emptiness*, that is the problem of checking whether the set contains any parameter valuations at all, and the *universality*, that is the problem of checking whether the set contains all the admissible parameter valuations.

In the following we denote with the term *S-set* either $S(\varphi)$ or $S(\mathcal{A}, \varphi)$, for some PMITL formula φ and some L/U automaton $\mathcal{A}$. Analogously, we denote with *V-set* either $V(\varphi)$ or $V(\mathcal{A}, \varphi)$.

4.4 The Concept of Polarity of Parameterized Temporal Operators

In this section we prove some results on the polarity of the parameterized operators which give further insights on our formalism and are useful for the results of the following sections.

4.4.1 Definition of polarity

The concept of *polarity* is semantic and is related to whether the space of the values for a parameter such that the formula is satisfied is upward or downward closed. A temporal operator of PMITL is *upward-closed* if the interval in its subscript has a parameter from U in one of its end-points. Analogously, a temporal operator of PMITL is *downward-closed* if the interval in its subscript has a parameter from L in one of its end-points. The meaning of these definitions is clarified by the following lemma.

We first introduce some notation. Let $\approx \in \{\leq, \geq\}$. Given any parameter valuations v and v', and a parameter z, with $\approx_z$ we denote the relation defined as: $v \approx_z v'$ iff $v(z) \approx v'(z)$ and $v(z') = v'(z')$ for any other parameter $z' \neq z$.

Lemma 6 *Let z be a parameter occurring in φ, α be a timed sequence, and v and v' be parameter valuations.*

1. *For $z \in U$ and $v \leq_z v'$, if $(\alpha, v) \models \varphi$ then $(\alpha, v') \models \varphi$.*
2. *For $z \in L$ and $v \geq_z v'$, if $(\alpha, v) \models \varphi$ then $(\alpha, v') \models \varphi$.*

Proof Consider first part (1). Let $z \in U$ be a parameter occurring in φ and α be a timed sequence such that $(\alpha, v) \models \varphi$. We show that for all $t \geq 0$ and for all subformulas ψ of φ, it holds that $(\alpha, v, t) \models \psi$ implies $(\alpha, v', t) \models \psi$. By contradiction, assume that ψ is a *minimal* subformula of φ such that the assertion (1) does not hold (where "minimal" means that for all its subformulas the assertion (1) holds) and in particular assume that $(\alpha, v, t) \models \psi$ holds and $(\alpha, v', t) \not\models \psi$. We only consider in detail the cases where ψ is of the form $\psi' \,\mathcal{U}_I\, \psi''$ or $\psi' \,\mathcal{R}_I\, \psi''$, and I is a parameterized interval over the parameter z. In all the other cases, the hypothesis that ψ is minimal such formula is trivially contradicted.

In the first case, I is of the form $(c, d+z)$. From $(\alpha, v, t) \models \psi$, we get that there is a $t' \in I_v + t$ such that $(\alpha, v, t') \models \psi''$ holds, and $(\alpha, v, t'') \models \psi'$ holds for all $t < t'' < t'$. By our assumption, i.e., ψ is minimal subformula contradicting the stated property, we get that $(\alpha, v', t') \models \psi''$ holds, and $(\alpha, v', t'') \models \psi'$ also holds for all $t < t'' < t'$. Since $I_v \subseteq I_{v'}$, we clearly get that $(\alpha, v', t) \models \psi$, thus contradicting the hypothesis.

In the second case, I is of the form $(c+z, d)$. From $(\alpha, v, t) \models \psi$, we get that either for all $t' \in I_v + t$, $(\alpha, v, t') \models \psi''$ holds, or there is a $t'' > t$ such that $(\alpha, v, t'') \models \psi'$

holds and $(\alpha, v, t') \models \psi''$ holds for all $t' \in [t, t''] \cap I_v + t$. Note that now $I_{v'} \subseteq I_v$. Therefore, if $(\alpha, v, t') \models \psi''$ holds for all $t' \in I_v + t$, then by our assumption, also $(\alpha, v', t') \models \psi''$ holds for all $t' \in I_{v'} + t$. In the other case, i.e., there is a $t'' > t$ such that $(\alpha, v, t'') \models \psi'$ holds and $(\alpha, v, t') \models \psi''$ holds for all $t' \in [t, t''] \cap I_v + t$, again by our assumption, also $(\alpha, v', t'') \models \psi'$ holds and $(\alpha, v', t') \models \psi''$ holds for all $t' \in [t, t''] \cap I_{v'} + t$.

Therefore, part (1) of the lemma is shown. Part (2) can be shown using similar arguments, and thus we omit the details. □

The restrictions imposed on the use of the parameters guarantee the following properties. Let z be a parameter occurring in φ and α be a timed sequence. If $z \in U$, then for each admissible valuation v, if $(\alpha, v, 0) \models \varphi$ then $(\alpha, v', 0) \models \varphi$ as well, for any v' such that $v \leq_z v'$. Thus in this case we say that the parameterized operator is *upward-closed*. Analogously, if $z \in L$, the parameterized operator is said *downward-closed*, since for each admissible valuation v, if $(\alpha, v, 0) \models \varphi$ then $(\alpha, v', 0) \models \varphi$ as well, for any v' such that $v \geq_z v'$.

4.5 Practical Use of PMITL

PMITL formulas can express meaningful properties of real-time systems. As an example, we consider a model of the SPSMALL memory, a commercial product of STMicroeletronics, analyzed by Chevallier et al. [2009]. Then, we show interesting properties that can be expressed using our formalism.

4.5.1 Model of a wire component in a memory circuit

The designer of a memory circuit guarantees the correct behavior of its product and some quantitative performance, called "(guaranteed) response times", assuming that the circuit is embedded into an environment satisfying some quantitative requirements, called "(external) requirements" [Chevallier et al., 2009].

Usually, the response times represent an upper bound on the time taken by the circuit to produce the result of a (write or read) operation whereas the external requirements specify the lower bounds on the clock cycle times and the stability times of

input signals. These values of the parameters of the specification form the datasheet of the circuit provided by the manufacturer to the customer and, generally, they are determined by electrical simulation.

To perform such a simulation, each component of the memory is modelled at the transistor level as a set of differential equations, which represent the Kirchoff laws associated with the electric current traversing the component; these differential equations depend on the physical characteristics of the transistors and wires (capacitors, resistors, . . .). The full memory is thus represented as a big system of differential equations. However, such a simulation process is much too long to be performed in a complete manner. Therefore sensitive portions of the circuit, which are supposed to contain the longest paths of traversal, are identified by hand and electrical simulations are performed only for such limited portions of circuit, which are assumed to contain the critical paths, but such an assumption of criticality is risky: it is very difficult to identify by hand relevant sensitive portions of the circuit, especially when the complexity of the circuit increases.

The need for formal methods to verify the timing values of the datasheet is therefore widely recognized. The memory is modeled as the synchronous product of the timed automata corresponding to the input signals and the internal components of the memory, such as latches, wires and logical blocks. Clarisó and Cortadella [2004a,b] propose a formal method for verifying the timings of asynchronous circuits. Their approach infers a set of sufficient linear constraints relating the delays of the internal gates of the circuit to the external delays of the circuit specification that guarantee the correct behavior of the circuit. The method is based on the reachability analysis of a timed model of the circuit (with additional abstract interpretation techniques [Cousot and Cousot, 1977].

As pointed out by Clarisó and Cortadella [2004b] , such parametric constraint sets are very informative for the designer, as they identify sensitive parts of the circuits (e.g., critical paths) and interrelations between various data of the specification. Moreover, many technology mappings can be tested immediately (by mere instantiation of the parameters).

Chevallier et al. [2009] follow a similar approach for formally verifying some generic properties of a commercial memory designed by STMicroelectronics, called SPSMALL. Their timing analysis method derives a set of sufficient linear constraints

relating the external parameters of the datasheet to the internal gate delays that guarantee the correctness of the circuits behavior. In particular, these constraints can be seen as sufficient conditions for certain paths of the circuit to be critical (i.e. those along which the propagation delay is the longest). Using the model of parametric timed automata (see [Alur and Dill, 1994]) and tool HYTECH [Henzinger et al., 1995] for reachability analysis, they are able to generate a set of linear constraints that ensures the correctness of some crucial timing behaviors of the memory.

In Figure 4.2, we recall the model for a wire. Observe that differently from the work of Chevallier et al. [2009], our model is a PTA where the system constants $x^{\uparrow}$, $x^{\downarrow}$, $y^{\uparrow}$, and $y^{\downarrow}$ are parameters. In particular, $x^{\uparrow}$ and $x^{\downarrow}$ are upward parameters (i.e., belong to U) and $y^{\uparrow}$ and $y^{\downarrow}$ are downward parameters (i.e., belong to L). The behavior of the wire is related to the intervals $[y^{\uparrow}, x^{\uparrow}]$ and $[y^{\downarrow}, x^{\downarrow}]$. These intervals represent the delay interval for the component traversal when the input signal is respectively rising and falling. The model has one clock variable z_0 and five locations. The symbol $r^{\uparrow}$ (resp. $r^{\downarrow}$) labels the location which is entered when the input signal r is rising (resp. falling), and similarly $o^{\uparrow}$ (resp. $o^{\downarrow}$) for the output signal o. Each edge corresponds to a discrete event in the system. The locations are associated with parametric clock constraints modeling the desired behaviour.

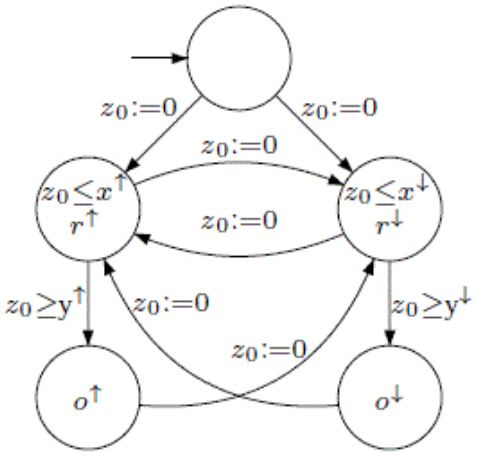

Figure 4.2: *A* PTA *model for the wire component.*

4.5.2 Properties expressed by PMITL formulas

When the data-sheet of a circuit does not provide enough information, we can use parameters to formulate the wished system requirements and then perform a parameterized analysis of the circuit. A crucial step in such analysis consists of solving the introduced decision problems.

As sample properties that can be expressed by PMITL formulas, consider

$\varphi_1 = \Box(r^{\uparrow} \Rightarrow \Diamond_{[c,d+x]} o^{\uparrow})$ and $\varphi_2 = \Box(r^{\downarrow} \Rightarrow \Diamond_{[c+y,d]} o^{\downarrow})$,

where $c, d \in \mathbb{N}$, $x \in U$ and $y \in L$. These are variations with parameters of the standard time response property. In particular, φ_1 asserts that "every rising edge of the input signal r is followed by a rising edge of the output signal o within the interval $[c, d + x]$". In the first formula (which is a PMITL$_{\Diamond}$ formula), we fix a constant lower bound on the response property and restrict the upper bound to be at least d. In the second formula instead, we fix an upper bound d and require the lower bound to be at least c.

We can use the logic PMITL in order to express some properties when we have partial informations in the description of the datasheet of the circuit. For example, suppose that we only know the upper bound of the wire traversal delay and we do not know the exact value of the lower bound, so we can ask if there exists an admissible value for the lower bound. We can verify this condition through the following property:

$\varphi_2 = \Box(r^{\uparrow} \Rightarrow \Diamond_{[c+x_1,d]} o^{\uparrow}) \wedge \Box(r^{\downarrow} \Rightarrow \Diamond_{[c+x_2,d]} o^{\downarrow})$,

where $c, d \in \mathbb{N}$ and $x_1, x_2 \in U$. The value c may be established by the mechanical specifics of the wire component and it is the minimum possible lower bound. In the following, we will see how to solve the satisfiability problem for φ_1 and φ_2 in order to know if there exists a valuation for the parameters which satisfy the given formula.

Moreover, we can extend this reasoning to other kind of problems, for example we can consider the case of asynchronous pipeline. Suppose our scope is to guarantee the synchronization between the environment and the pipeline assuring that whenever the environment sends new data to the pipeline, then we will have a following production of data after that the first stage of the pipeline is empty. We can express this property with the logic PMITL :

$\varphi_3 = \Box(prod \Rightarrow \neg prod \, \mathcal{U}_{[c,d+x]} \, empty(S_1))$,

where $c, d \in \mathbb{N}$, $x \in U$, *prod* represents the production of new data by the environ-

ment and S_1 is the first stage of the pipeline.

We may verify if there exists a valuation for the parameter x which satisfy φ_3 and assure a correct synchronization between the environment and the pipeline.

4.6 Preliminary Results

In this section we prove some results on the duality of the parameterized operators. We also present a transformation of PMITL formulas that shifts the admissible domain for each parameter not occurring in constraints of the form $(c + z, d)$, to match the whole set $\mathbb{N}$.

4.6.1 Negation normal form for PMITL formulas

The formulas of PMITL are in *negation normal form*, that is negation occurs only at the level of the atomic propositions. This is standard for parameterized linear temporal logic. In fact, the negation inverts the polarity of the parameterized temporal operators occurring within the scope of the negation operator. Nevertheless, our logic is closed under semantic negation.

For a PMITL formula φ, we denote with $\sim\varphi$ the PMITL formula obtained from φ by replacing each operator with its dual (i.e., exchanging the $\vee$'s with the $\wedge$'s and the $\mathcal{U}$'s with $\mathcal{R}$') and negating the atomic propositions.

To show that $\sim\varphi$ is semantically equivalent to the negation of φ, consider the following proposition which states a noteworthy equivalence that establishes the duality of the temporal operators of PMITL. The proof of this equivalence can be directly derived from the semantics of the operators.

Proposition 7 *For any interval I:* $\quad \neg(\varphi\,\mathcal{U}_I\,\psi) \equiv \neg\varphi\,\mathcal{R}_I\,\neg\psi.$

By the above proposition and the De Morgan's laws, it is simple to see the following:

Proposition 8 *For any PMITL formula φ:* $\quad \neg\,\varphi \equiv \sim\varphi.$

Observe that if I is a parameterized interval, then the operators $\mathcal{U}_I$ and $\mathcal{R}_I$ have opposite polarities, and then in $\sim\varphi$ the role of the sets L and U is exchanged (i.e., $\sim\varphi$

is a PMITL formula over the set of lower bound parameters U and the upper bound parameters L), and in particular the following holds.

Proposition 9 *For any PMITL formula φ, φ is a PMITL$_\Diamond$ formula if and only if $\sim\varphi$ is a PMITL$_\Box$ formula.*

Finally, the satisfiability of $\sim\varphi$ is dual with respect to the validity of φ. In fact, one can observe that φ and $\sim\varphi$ have the same set of admissible parameter valuations and $V(\varphi)$ is the complement of $S(\sim\varphi)$, with respect to set of the admissible parameter valuations, and analogously, $V(\mathcal{A}, \varphi)$ is the complement of $S(\mathcal{A}, \sim\varphi)$. Thus, we get the following lemma.

Lemma 10 *The emptiness/universality problems of V-sets for a PMITL formula φ reduce in linear time to the universality/emptiness problems of S-sets for $\sim\varphi$, and viceversa.*

Proof As observed, $\mathcal{D}(\sim\varphi) = \mathcal{D}(\varphi)$, since the subscripted intervals in φ and $\sim\varphi$ are the same. Since $V(\varphi) = \mathcal{D}(\varphi) \setminus S(\sim\varphi)$, then $V(\varphi) = \varnothing$ if and only if $S(\sim\varphi) = \mathcal{D}(\sim\varphi)$, and $V(\varphi) = \mathcal{D}(\varphi)$ if and only if $S(\sim\varphi) = \varnothing$.

Analogously, $V(\mathcal{A}, \varphi) = \mathcal{D}(\varphi)$ if and only if $S(\mathcal{A}, \sim\varphi) = \varnothing$, and $V(\mathcal{A}, \varphi) = \varnothing$ if and only if $S(\mathcal{A}, \sim\varphi) = \mathcal{D}(\sim\varphi)$.

Clearly, the formula $\sim\varphi$ has the same size as φ, and thus the statement follows. □

4.6.2 Restrictions on the parameters

In order to obtain decidability results, the need for restricting the use of parameters, such that each parameter is always used with a fixed polarity, was addressed already by Alur et al. [2001] for obtaining the decidability of PLTL. Both in the logic PLTL [Alur et al., 2001] and in $\mathrm{P}_{0,\infty}\mathrm{MITL}_{0,\infty}$ [Bozzelli and La Torre, 2009], time constraints on temporal operators do not allow intervals with arbitrary end-points: one of the end-points is always 0 or ∞. The techniques used by Alur et al. [2001] and by Bozzelli and La Torre [2009] to show decidability results, cannot be used directly in our settings.

Observe that we have defined PMITL by imposing some restrictions on the parameters. First, we require the sets of parameters L and U to be disjoint. Second, we force each interval to have at most one parameter, either in the left or in the right end-point.

Third, we define admissibility for parameter valuations such that a parameterized interval cannot be evaluated neither as an empty nor a singular set.

In the Chapter 5, we will discuss the impact of the first two restrictions on the decidability of the considered problems. In particular, we show that relaxing any of the first two restrictions leads to undecidability. Concerning to the notion of admissibility of parameter valuations, the restriction to non-empty intervals seems reasonable since evaluating an interval to an empty set would cancel a portion of our specification (which would remain unchecked). Finally, the restriction to non-singular sets, which equals to disallow equality in the constraints, is already present in the temporal logic MITL [Alur et al., 1996], the parameterized discrete-time logic PLTL [Alur et al., 2001] and the parametric timed automata [Bozzelli and La Torre, 2009; Hune et al., 2002], and in all these formalisms it is crucial for achieving decidability. The same arguments used there can be applied here to show undecidability of the decision problems for PMITL if this restriction is relaxed.

4.6.3 Normalization of intervals

A formula containing intervals of the form $(c + z, d)$ with $c \geq d$ would not have admissible parameter valuations, and the considered decision problems become trivial. Therefore, in our proofs, we need to argue only the cases when $c < d$ for such intervals. The situation is quite different for parameterized intervals of the form $(c, d + z)$. In fact, in this case, there would be admissible valuations for any natural numbers c and d. However, it is often convenient to restrict to formulas where $c < d$.

A PMITL formula φ is *well defined* if $c < d$ for all its parameterized intervals of the form $(c, d + z)$. Observe that, if the formula is well defined, and z is one of its parameters not occurring in any constraint of the form $(c + z, d)$, then each non negative number can be assigned to z by an admissible valuation. The following lemma shows that it suffices to solve the introduced decision problems for well defined PMITL formulas.

For parameter valuations v, v', we denote with $v - v'$ the function that maps each parameter z to $v(z) - v'(z)$.

Lemma 11 *Given a PMITL formula φ, there exists a normalized PMITL formula φ' such that $N_{\varphi'} = N_\varphi$, $K_{\varphi'} = O(K_\varphi)$, and there exists a parameter valuation v' such*

that:

- *for each parameter valuation* v*:* $v \in \mathcal{D}(\varphi)$ *if and only if* $v - v' \in \mathcal{D}(\varphi')$*,*
- *for each timed sequence* α*:* $(\alpha, v) \models \varphi$ *if and only if* $(\alpha, v - v') \models \varphi'$*.*

Proof Fix a PMITL formula φ. For each parameter z appearing in φ, define the constant m_z as:

$$m_z = \begin{cases} max\{c - d + 1 | (c, d + z) \text{ is in } \varphi\} & \text{if } max\{c - d + 1 | (c, d + z) \text{ is in } \varphi\} > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Let φ' be the PMITL formula obtained from φ by replacing each occurrence of a parameter z with $m_z + z$. Thus, each interval of the form $(c, d + z)$ is replaced with $(c, d + m_z + z)$, and each interval of the form $(a + z, b)$ with $(a + m_z + z, b)$. Clearly, from the definition of the constants m_z, we get that φ' is well defined.

Denote with v' the parameter valuation that maps each parameter z to m_z. From the definition of admissibility, if a parameter valuation v belongs to $\mathcal{D}(\varphi)$, then $v(z) \geq m_z$ for each parameter z, and thus $v - v'$ is a parameter valuation (i.e., it assigns a natural number to each parameter).

Observe that an admissibility constraint $v(z) > c - d$, which derives from an interval $I = (c, d + z)$ in φ, is replaced with $v'(z) > c - d - m_z$, from the interval $I' = (c, d + m_z + z)$ in φ'. Analogously, an admissibility constraint $v(z) < b - a$, deriving from $I = (a + z, b)$ in φ, is replaced with $v'(z) < b - a - m_z$ from $I' = (a + m_z + z, b)$ in φ'. Thus, $v - v' \in \mathcal{D}(\varphi')$ if and only if $v \in \mathcal{D}(\varphi)$. Moreover, for any choice of I and I' as above, $I_v = I'_{v'}$ holds. Thus, since both or none of v and $v - v'$ are admissible and they evaluate corresponding parameterized intervals of φ and φ' to the same time interval, we have that α satisfies φ under valuation v if and only if α satisfies φ' under valuation $v - v'$.

Now, let M be the maximum value of the defined m_z. Clearly, there is a constant c in the formula φ such that M is less than or equal to $c + 1$, and then $M \leq K_\varphi$ (recall that K_φ is the maximal constant used in φ augmented by 1). Moreover, the constants in φ' exceed those in φ of at most M, therefore $K'_\varphi \leq 2K_\varphi$ holds.

Finally, φ' has the same number of the subformulas as φ, and this completes the proof. □

4.6.4 Expressiveness: comparing PMITL vs. MITL.

The logic PMITL adds to MITL in terms of expressiveness, in the sense that some properties can be expressed in PMITL but not in MITL.

To see this, given a PMITL formula φ, denote with $L_{exist}(\varphi)$ the set of timed sequences $\{\alpha \mid \exists v \text{ such that } (\alpha, v) \models \varphi\}$. The following proposition holds.

Proposition 12 *Let φ be the PMITL formula $\Box(\neg p \Rightarrow \Diamond_{]0,x]} p)$, for $x \in U$ and an atomic proposition p. There is no* MITL *formula φ' such that the set of all the timed words satisfying φ' is exactly $L_{exist}(\varphi)$.*

Proof Observe that the set $L_{exist}(\varphi)$ contains all the timed sequences α such that there is a bound $b_\alpha \in \mathbb{N}$ on the length of all the intervals where $\neg p$ holds. Now, let φ' be an MITL formula which is satisfied by all the timed sequences of $L_{exist}(\varphi)$ (if such a formula does not exists we are done). For $m \in \mathbb{N}$, let α_m be the timed sequence such that the proposition p holds only at all the times nm, for $n \in \mathbb{N}$. Clearly, $\alpha_m \in L_{exist}(\varphi)$.

For an MITL formula ψ, denote with *Closure*(ψ) the set containing all the subformulas of ψ and, for each subformula of the form $\psi' O_{(c,d)} \psi''$, for O being either $\mathcal{U}$ or $\mathcal{R}$, all the formulas $\psi' O_{(c-b,d-b)} \psi''$, for a non negative $b \leq c$, and all the formulas $\psi' O_{(0,a)} \psi''$, for $0 < a < d - c$. Given a timed sequence α, we can label each time t on α with all the formulas of *Closure*(ψ) which are fulfilled by the suffix of α starting at t. We call a *cycle* with respect to ψ of α any bounded portion of α from a time t to time t' such that $t' - t > 1$, and t and t' are labeled with the same set of formulas from *Closure*(ψ). Denoting with α' a timed sequence obtained from α by iterating an arbitrary number of times a cycle with respect to ψ, by induction on the structure of MITL formulas, one can show that if α satisfies ψ, then also α' satisfies ψ.

Now, take $m > 2^{N_{\varphi'} K_{\varphi'}}$. Since the number of formulas in *Closure*(φ') is bounded by $N_{\varphi'} K_{\varphi'}$, we are guaranteed that in each portion of α_m between two consecutive occurrences of p, there is a cycle with respect to φ'. Therefore, we can iterate such cycles such that in the resulting sequence β the distance between two consecutive

occurrences of p grows unboundedly. Clearly, β does not belong to $L_{exist}(\varphi)$ but fulfills φ'. Therefore, there are no MITL formulas that can express φ. □

4.7 Decidability of PMITL

In this section, we prove that the satisfiability, validity, and model-checking problems defined in the section 4.3 are decidable and EXPSPACE-complete, thus matching the computational complexity for MITL formulas [Alur et al., 1996]. A central step in our argument is a translation to the emptiness and the universality problems for Büchi L/U automata.

4.7.1 Normal form and equivalences for PMITL formulas

First we show a normal form for well defined PMITL formulas. A PMITL formula is in *normal form* if each of its subformulas is of one of the following types:

1. $\Diamond_{]0,b[}\ \varphi'$, or $\Diamond_{]0,b]}\ \varphi'$, where $b \in \mathbb{N}$;
2. $\Box_{]0,b[}\ \varphi'$, or $\Box_{]0,b]}\ \varphi'$, where $b \in \mathbb{N}$;
3. $\varphi_1\ \mathcal{U}_{(a,b)}\ \varphi_2$, or $\varphi_1\ \mathcal{R}_{(a,b)}\ \varphi_2$, where $a, b \in \mathbb{N}$ and $a > 0$;
4. $\varphi_1\ \mathcal{U}_{]0,\infty[}\ \varphi_2$;
5. $\Box_{]0,\infty[}\ \varphi'$;
6. $\Diamond_{(0,x)}\varphi'$, where $x \in U$;
7. $\Box_{(0,y)}\ \varphi'$, where $y \in L$;
8. $\Box_{(a+x,b)}$, where $a, b \in \mathbb{N}$, $x \in U$;
9. $\Diamond_{(a+y,b)}$, where $a, b \in \mathbb{N}$, $y \in L$.

We start showing the following lemma.

Lemma 13 *For every well defined PMITL formula φ, there is an equivalent PMITL formula ψ using only $\Diamond$ and $\Box$ as parameterized temporal operators, and using only parameterized intervals of the form $(0, z)$ or $(c + z, d)$, for $c, d \in \mathbb{N}$ and $z \in U \cup L$. Moreover, $N_\psi = O(N_\varphi)$ and $K_\psi = K_\varphi$.*

Proof The first step consists of showing that the parameterized operators $\Diamond_H$ and $\Box_J$ are sufficient to define all the parameterized operators of PMITL. This can be achieved by rewriting a well defined PMITL formula according to the following equivalences:

1. $\varphi \,\mathcal{U}_{(c,d+z)}\, \psi \equiv (\varphi \,\mathcal{U}_I\, \psi) \wedge \Diamond_{(c,d+z)}\psi$,
2. $\varphi \,\mathcal{R}_{(c,d+z)}\, \psi \equiv (\varphi \,\mathcal{R}_I\, \psi) \vee \Box_{(c,d+z)}\, \psi$,
3. $\varphi \,\mathcal{U}_{(c+z,d)}\, \psi \equiv \Box_{[0,z]}(\varphi \,\mathcal{U}_I\, \psi) \wedge \Diamond_{(c+z,d)}\psi$,
4. $\varphi \,\mathcal{R}_{(c+z,d)}\, \psi \equiv \Diamond_{[0,z]}(\varphi \,\mathcal{R}_I\, \psi) \vee \Box_{(c+z,d)}\, \psi$,

where $I = (c, \infty[$ and: for the equivalences 1 and 2, I is left-closed if and only if $(c, d + z)$ is left-closed, and for the equivalences 3 and 4, I is left-closed iff $(c + z, d)$ is left-closed.

By Proposition 7, the second equivalence can be obtained from the first one and the fourth one from the third one by duality.

To see the equivalence 1, observe that by definition, $\varphi \,\mathcal{U}_{(c,d+z)}\, \psi$ is satisfied on a timed sequence α under a valuation v if and only if there is a $t \in (c, d + v(z))$ such that $(\alpha, v, t) \models \psi$ and for all positive $t' < t$, $(\alpha, v, t') \models \varphi$. Moreover, this second assertion is equivalent to require that both: (1) there is a $t \geq c$ (or $t > c$, if I is left-open) such that $(\alpha, v, t) \models \psi$ and for all positive $t' < t$, $(\alpha, v, t') \models \varphi$, and (2) there is a $t \in (c, d + v(z))$ such that $(\alpha, v, t) \models \psi$. But this is equivalent to require that the conjunction of $\varphi \,\mathcal{U}_I\, \psi$ and $\Diamond_{(c,d+z)}\psi$ must hold on α under v, and thus the equivalence holds.

For the equivalence 3, suppose that $(\alpha, v) \models \varphi \,\mathcal{U}_{(c+z,d)}\, \psi$. From the semantics, clearly $(\alpha, v) \models \Diamond_{(c+z,d)}\psi$ also holds. Moreover, there must be a $t \in (c+v(z), d)$ such that $(\alpha, v, t) \models \psi$ and for all positive $t' < t$, $(\alpha, v, t') \models \varphi$. From $t \in (c + v(z), d)$, we get that $t - t'' \in I$ for all $t'' \in [0, v(z)]$, and thus also $(\alpha, v) \models \Box_{[0,z]}(\varphi \,\mathcal{U}_I\, \psi)$ holds. For the converse direction, from $(\alpha, v) \models \Box_{[0,z]}(\varphi \,\mathcal{U}_I\, \psi)$, we get that there is a $t \in I$ such that $(\alpha, v, v(z) + t) \models \psi$ and $(\alpha, v, t') \models \varphi$ for each positive $t' < t$.

Since $v(z) + t \ (c + v(z), \infty[$ and $(\alpha, v) \models \Diamond_{(c+z,d)}\psi$ also holds, we can assume that $v(z) + t \in (c + v(z), d)$ and thus, $(\alpha, v) \models \varphi \, \mathcal{U}_{(c+z,d)} \, \psi$.

Note that the parameters in all the above equations are used with the same polarity and without changing the set of admissible values.

The second step consists of removing all the intervals of the $(c, d + z)$ from the operators $\Box$ and $\Diamond$. Note that by hypothesis the formula is well defined, and therefore we can safely use intervals of the form (c, d) and $[0, z]$ to do so. This step can be achieved using the following equivalences which can be derived directly from the given semantics:

1. $\Diamond_{(c,d+z)}\varphi \equiv \Diamond_{(c,d)}\Diamond_{[0,z]}\varphi$,
2. $\Box_{(c,d+z)} \, \varphi \equiv \Box_{(c,d)} \, \Box_{[0,z]} \, \varphi$,

where (c, d) is left/right open if $(c, d + x)$ is left/right open. Again, the polarity of the parameters in the above equivalences is the same on both sides of the same equivalence.

Observe that the transformations described by the above equivalences keep unchanged the maximal constant of the formula and introduce at most a constant number of new subformulas for each subformula of the starting formula. Therefore, we can rewrite a PMITL formula φ into an equivalent formula ψ such that $K_\psi = K_\varphi$ and $N_\psi = O(N_\varphi)$ and thus the lemma holds. $\Box$

4.7.2 Construction of L/U automaton

By Lemma 13, we can assume that PMITL formulas are in normal form with respect to all the parameterized operators. Recall that in the work of Alur et al. [1996], it is shown how to reduce in normal form each MITL formula. Observe that such transformations can be applied to PMITL formulas without altering the polarity of the parameterized temporal operators. Therefore we can apply them to the PMITL formulas resulting from Lemma 13, thus obtaining an equivalent PMITL formula in normal form.

Lemma 14 *Given a well defined PMITL formula φ, there is an equivalent PMITL formula ψ in normal form such that $N_\psi = O(N_\varphi)$ and $K_\psi = K_\varphi$.*

If we restrict to PMITL formulas which do not contain parametric intervals of the form $(c + z, d)$, we are able to adapt the constructions given by Alur et al. [1996]

for MITL and by Bozzelli and La Torre [2009] for $P_{0,\infty}MITL_{0,\infty}$ to obtain equivalent PTAs.

This result is precisely stated in the following theorem.

Theorem 15 *Given a PMITL formula φ in normal form, which does not contain intervals of the form $(c+z, d)$, one can construct a Büchi L/U automaton $\mathcal{A}_\varphi$ such that $\mathcal{A}_\varphi$ accepts a timed sequence α under a parameter valuation v if and only if $(\alpha, v) \models \varphi$. Also, $\mathcal{A}_\varphi$ has $O(2^{N_\varphi \times K_\varphi})$ locations, $O(N_\varphi \times K_\varphi)$ clocks, and constants bounded by K_φ. Moreover, if φ has only parameters from the set L (resp. U) the resulting automaton $\mathcal{A}_\varphi$ also has only lower bound parameters from L (resp. upper bound parameters from U).*

Proof Fix a PMITL formula φ. By the results of Alur et al. [1996], for each parameter valuation v and timed sequence α, there is a timed sequence $\alpha_{\varphi,v}$ which is equivalent to α such that for each interval I of $\alpha_{\varphi,v}$ and subformula ψ of φ, the following holds: for all $t, t' \in I$, $(\alpha, v, t) \models \psi$ iff $(\alpha, v, t') \models \psi$. Thus, for a given parameter valuation v, we can restrict ourselves to consider only such timed sequences.

Observe that the construction given by Alur et al. [1996] defines the behaviors of the automaton modularly to each subformula of the input formula. Therefore, we only need to augment that construction with the portion concerning with the subformulas corresponding to the parameterized operators.

Since φ is in normal form, the only subformulas of φ corresponding to parameterized operators are of the form $\Diamond_{(0,x)}\chi$ and $\Box_{(0,y)}\chi$.

We fix a parameter valuation v and consider first a subformula ψ of φ of the form $\Diamond_I\chi$, for $I = (0, x)$. To check the fulfillment of ψ, we need to use one clock which we call ξ, along with the clock constraint $\xi \in I$. As in [Alur et al., 1996], each location of $\mathcal{A}$ keeps track of the set of subformulas of φ which hold at the current time (w.r.t. the parameter valuation v). In order to witness the fulfillment of ψ at the current time t, the automaton resets ξ and stores in its finite control the obligation that χ must hold at a time $t + d$, such that $d \in I_v$. The obligation is discharged as soon as an appropriate state is found, keeping track that χ holds at the current time: in this case we have that ψ is fulfilled. We cannot reset ξ if an obligation for ψ is already pending and cannot be discharged, otherwise we would not be able to check the pending obligation. However, this is not needed since a witness for the

previous obligation will also prove the fulfillment of ψ at the current time. Once the obligation is discharged, the clock ξ can be reused. Thus, one clock suffices to check the subformula ψ as often as necessary. Moreover, each transition whose source is a location q that stores an obligation for ψ (that cannot be discharged locally), uses an atomic clock constraint $\xi \in I$ as a conjunct of the associated clock constraint. This ensures that an obligation for ψ is not mistakenly discharged because of a witness at a time $t + d$ with $d \geq v(x)$ (or $d > v(x)$). The behavior we have described needs to be implemented to check the truth of ψ over intervals and there are some subtleties concerning the treatment of open intervals that have been omitted here because the details on these aspects do not differ from the case of formulas of the form $\Diamond_{(0,b)}\chi$, where b is a constant, which is carefully explained by Alur et al. [1996].

A subformula ψ of φ of the form $\Box_{(0,y)}\,\chi$ is dual to formula of the form $\Diamond_{(0,x)}\chi$. Therefore, we can argue similarly that an automaton can check for the fulfillment of ψ, simply using a clock ξ and a clock constraint $\xi > y$, if the interval is right-closed, and $\xi \geq y$ if the interval is right-open.

From the work of Alur et al. [1996], we get that the non-parameterized portion of the construction of $\mathcal{A}_\varphi$ has $O(2^{N_\varphi \times K_\varphi})$ locations, $O(N_\varphi \times K_\varphi)$ clocks, and constants bounded by K_φ. Clearly, the addition we have described above does not alter these measures.

To complete the proof, observe that, the constructed $\mathcal{A}_\varphi$ uses exactly the parameters of the formula φ such that each parameter of φ from L is a lower bound parameter for $\mathcal{A}_\varphi$ and each parameter of φ from U is an upper bound parameter for $\mathcal{A}_\varphi$. □

4.7.3 Computational complexity

We can now show the main theorem of this Chapter, regarding the complexity results of PMITL.

Theorem 16 *The problems of checking the emptiness and the universality of the sets $S(\varphi)$ and $S(\mathcal{A}, \varphi)$, for any PMITL formula φ and any L/U automaton $\mathcal{A}$, are* EXPSPACE*-complete.*

Proof Hardness follows from EXPSPACE-hardness of both satisfiability and model-checking problems for MITL formulas [Alur et al., 1996].

By Lemma 11 we can consider a well defined formula. Moreover, by Lemma 14, we can transform a given well defined formula into an equivalent one in normal form and this can be done in linear time. Thus, to show the theorem it suffices to focus on solving the considered decision problems for normal form PMITL formulas and arguing that the proposed algorithms take at most exponential space.

Fix a normal form PMITL formula φ. We start with the emptiness problem for $S(\varphi)$.

We observe that if φ contains a parameterized interval of the form $(c+z, d)$ where $c \geq d$ then $\mathcal{D}(\varphi)$ is empty and thus $S(\varphi) = \varnothing$. In the remaining case, we use the following algorithm.

First, assign each parameter $x \in L$ appearing in a subformula of φ of the form $\Diamond_{(c+x,d)}\psi$, $c < d$, with the minimum value assigned by an admissible parameter valuation, and each parameter $y \in U$ in a subformula of φ of the form $\Box_{(c'+y,d')}\psi'$, $c' < d'$, with the maximum value assigned by an admissible parameter valuation. Note that these values are well defined since for such parameters the admissible values are bounded (in fact, from the first kind of formula we get the constraint $0 \leq x \leq d-c-1$, and for the second the constraint $0 \leq y \leq d' - c' - 1$). Now, denote with φ' the resulting formula, construct the Büchi L/U automaton $\mathcal{A}_{\varphi'}$ as in Theorem 18 and then check $\Gamma(\mathcal{A}_{\varphi'})$ for emptiness.

Since to obtain φ' we assign with the minimum admissible value only parameters of L and with the maximum admissible value only parameters of U, by Lemma 6 we get that $S(\varphi)$ is empty iff $S(\varphi')$ is empty. Moreover, from the construction above, the number of subformulas of φ' is at most that of φ. Thus, by Theorems 5 and 18, we get that the above algorithm correctly checks $S(\varphi)$ for emptiness and takes exponential space.

For showing EXPSPACE membership of the universality problem for $S(\varphi)$ we can reason analogously. First, assign each parameter x appearing in a subformula of φ of the form $\Diamond_{(c+x,d)}\psi$ with the maximum value assigned by an admissible parameter valuation, and each parameter y in a subformula of φ of the form $\Box_{(c'+y,d')}\psi'$ with the minimum value assigned by an admissible parameter valuation.

Now, denote with φ' the resulting formula, construct the Büchi L/U automaton $\mathcal{A}_{\varphi'}$ as in Theorem 18 and then check $\Gamma(\mathcal{A}_{\varphi'})$ for universality.

Membership to EXPSPACE of deciding emptiness and universality of $S(\varphi, \mathcal{A})$ can

be shown with similar arguments, we just need to change the last step of the above algorithm to check the desired property of the set Γ for the intersection of the given $\mathcal{A}$ and the constructed $\mathcal{A}_{\varphi'}$, and not just for $\mathcal{A}_{\varphi'}$. □

By Theorem 19 and Lemma 10 the following result also holds.

Corollary 1 *The problems of checking the emptiness and the universality of the sets $V(\varphi)$ and $V(\mathcal{A},\varphi)$, for any PMITL formula φ and any L/U automaton $\mathcal{A}$, are* EXPSPACE*-complete.*

Chapter 5

Fragments and Extensions of PMITL

In this Chapter we focus on the study of the computational complexity of natural syntactic fragments of PMITL, showing that in meaningful fragments of the logic the considered decision problems are PSPACE-complete. Moreover, we consider a remarkable problem expressed by queries where the values that each parameter may assume are either existentially or universally quantified. We solve this problem in several cases and we propose an algorithm in EXPSPACE. Since we have defined the logic PMITL imposing some restrictions on the use of the parameters, we show, there, that if we relax any of the restrictions imposed, the decision problems become undecidable.

5.1 $\mathbf{P}_{0,\infty}\mathbf{MITL}_{0,\infty}$

The first work that solves verification problems against linear-time specifications with parameters both in the model and in the specification is presented by Bozzelli and La Torre [2009].

In this work, the authors investigate the class of L/U automata and consider acceptance conditions over infinite runs. Bozzelli and La Torre show that questions about the set $\Gamma(A)$ of parameter valuations, for which an L/U automaton A has an infinite accepting run, can be answered considering a bounded set of parameter valuations of size exponential in the size of the constants and the number of clocks, and polynomial in the number of parameters and locations of A. Therefore, they are able to show that checking the set $\Gamma(A)$ for emptiness, universality, and finiteness is PSPACE-complete.

The main argument for such results in their paper is given as follows: suppose that A is an L/U automaton which uses parameters only as lower bounds (resp., upper bounds); then if an infinite run ρ is accepted by A for large-enough values of the parameters, it is possible to determine appropriate finite portions of ρ which can be "repeatedly simulated" (resp., "deleted") thus obtaining a run ρ' which is accepted by A for larger (resp., smaller) parameters values.

As an extension of the above results, the authors consider constrained emptiness and constrained universality on L/U automata, where the constraint is represented by a linear system over parameters. They show that these problems are in general undecidable, and become decidable in polynomial space (and thus PSPACE-complete) if they do not compare parameters of different types in the linear constraints. Moreover, they show that when all the parameters in the model are of the same type (i.e., either lower bound or upper bound), it is possible to compute an explicit representation of the set $\Gamma(A)$ by linear systems over parameters whose size is doubly exponential in the number of parameters. An important consequence of their results on L/U automata is the extension to the dense-time paradigm of the results shown by Alur et al. [2001]. They define a parametric extension of the dense-time linear temporal logic $\text{MITL}_{0,\infty}$[Alur et al., 1996], denoted $\text{PMITL}_{0,\infty}$, and they show that (under restrictions on the use of parameters analogous to those imposed on L/U automata) the related satisfiability and model-checking problems are PSPACE-complete. The proof consists of translating formulas to L/U automata.

Note that in the work of Bozzelli and La Torre [2009], the acronym $\text{PMITL}_{0,\infty}$ is used to denote the parametric extension of $\text{MITL}_{0,\infty}$ where also the parameterized intervals are restricted such that one of the end-points is either 0 or ∞. Here, we prefer to denote such extension of $\text{MITL}_{0,\infty}$ as $\text{P}_{0,\infty}\text{MITL}_{0,\infty}$ to stress the fact that the imposed syntactic restriction concerns both time and parametric intervals.

Now, we consider an interesting example proposed by Bozzelli and La Torre [2009].

Example 1 *A typical property that can be stated in $P_{0,\infty}MITL_{0,\infty}$ is a parametric version of the usual time response property:*

$\phi = \Box(a \rightarrow \Diamond_{\leq l} b)$,

that is "whenever a holds true then b should holds true within time l", where l is a lower bound parameter.

Compared to the time response property, here it is not needed to specify a constant for the required delay, and it can instead used a parameter to express it. It is possible to perform the analysis for any possible value of the constant and leave to a later time the task of determining the exact constant. For example, consider the timed sequence

$\alpha = (\{a\}, [0, 2[)(\{b\}, [2, 4[)(\varnothing, [4, 5[)(\varnothing, [5; 6[) \ldots$

where all atomic propositions hold false starting from time 4. It is possible to determine that the set of parameter valuations v such that α satisfies ϕ under v is given by $l \geq 2$, and thus reply to several questions on the possible values of the parameter, such as what is the minimum constant for the property to hold.

The authors solve the considered decision problems by reducing them to corresponding problems on L/U automata. The key of these reductions is the translation of $\mathrm{P}_{0,\infty}\mathrm{MITL}_{0,\infty}$formulas into equivalent L/U automata.

5.1.1 Definition and known results

The logic $\mathrm{P}_{0,\infty}\mathrm{MITL}_{0,\infty}$ extends MITL [Alur et al., 1996] by allowing parameterized time intervals as subscripts of temporal operators.

The $\mathrm{P}_{0,\infty}\mathrm{MITL}_{0,\infty}$ formulas over AP are defined by the following grammar:

$$\varphi \;:=\; p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \,\mathcal{U}_E\, \varphi \mid \varphi \,\mathcal{R}_F\, \varphi,$$

where $p \in AP$, and E and F are either non-singular time intervals with one endpoint in $\mathbb{N}$ and the other one in $0 \cup \{\infty\}$ or parameterized time intervals such that:

- for $E = (a, b)$ either (1) $a = 0$ and $b \in \mathcal{E}(U)$ or (2) $a \in \mathcal{E}(L)$ and $b = \infty$;
- for $F = (a, b)$ either (1) $a = 0$ and $b \in \mathcal{E}(L)$ or (2) $a \in \mathcal{E}(U)$ and $b = \infty$.

As usual, we use the abbreviations $\Diamond_E \varphi$ and $\Box_F \varphi$ for $true\, \mathcal{U}_E\, \varphi$ and $false\ \mathcal{R}_F\ \varphi$, respectively.

$\mathrm{P}_{0,\infty}\mathrm{MITL}_{0,\infty}$ formulas are interpreted over timed sequences and with respect to a parameter valuation. For a formula φ, a timed sequence α, a parameter valuation v, and $t \in \mathbb{R}_+$, the *satisfaction relation under valuation* v, denoted $(\alpha, v, t) \models \varphi$, is defined as we have already seen for PMITL formulas in Section 4.3.

For the logic $\mathrm{P}_{0,\infty}\mathrm{MITL}_{0,\infty}$ Bozzelli and La Torre [2009], have studied the satisfiability, the validity and the model-checking problems with respect to an L/U automaton.

They solve the decision problems, defined in the same way of the logic PMITL described in Section 4.3, by reducing them to corresponding problems on L/U automata. The key of these reductions is the translation of $P_{0,\infty}MITL_{0,\infty}$ formulas into equivalent L/U automata. Such a translation relies on the result given by Alur et al. [1996] concerning $MITL_{0,\infty}$ and timed automata, which we recall.

Theorem 17 *([4] in [Alur et al., 1996]) Let φ a $MITL_{0,\infty}$ formula. Then one can construct a timed automaton $\mathcal{A}_\varphi$ such that $\mathcal{A}_\varphi$ accepts a timed sequence α if and only if α is a model of φ. Moreover, $\mathcal{A}_\varphi$ has $O(2^{N_\varphi})$ locations, $O(N_\varphi)$ clocks , and constants bounded by K_φ.*

Bozzelli and La Torre [2009] have proved the following result:

Theorem 18 *([15] in [Bozzelli and La Torre, 2009]) Given a $P_{0,\infty}MITL_{0,\infty}$ formula φ with lower (resp., upper) bound parameters in L (resp., U), one can construct an L/U automaton $\mathcal{A}_\varphi$ such that for each parameter valuation v: $(\mathcal{A}_\varphi)$ accepts a timed sequence α if and only if $(\alpha, v, o) \models \varphi$. Moreover, $\mathcal{A}_\varphi$ has $O(2^{N_\varphi})$ locations, $O(N_\varphi)$ clocks , and constants bounded by K_φ.*

We recall the main result of their paper:

Theorem 19 ***([16]**in [Bozzelli and La Torre, 2009]) The problems of checking the emptiness and the universality of the sets $S(\varphi)$ and $S(\mathcal{A}, \varphi)$, for any $P_{0,\infty}MITL_{0,\infty}$ formula φ and any L/U automaton $\mathcal{A}$, are* PSPACE*-complete.*

5.2 $PMITL_{0,\infty}$, $PMITL_{\Diamond}$ and $PMITL_{\Box}$

In this section, we address the complexity of the parameterized operators in PMITL, and thus of the corresponding logic fragments. Since MITL is already EXPSPACE-hard, we focus only on the fragments of $PMITL_{0,\infty}$ which include $MITL_{0,\infty}$. For this fragment of MITL it is already known that the satisfiability and the model-checking problems are PSPACE-hard.

On the positive side, we prove that the problems of checking the emptiness for the S-sets and the V-sets of $PMITL_{\Diamond}$ formulas are both in PSPACE, such as checking the dual universality problems for the S-sets and the V-sets of $PMITL_{\Box}$ formulas,

thus matching the lower bound known from their fragment $\mathsf{MITL}_{0,\infty}$. We complete our analysis by showing EXPSPACE-hardness for the remaining problems in $\text{PMITL}_{\Diamond}$ and $\text{PMITL}_{\Box}$, and all the considered problems in the fragments of PMITL allowing subformulas either of the form $\Box_{(c+x,d)}$ or of the form $\Diamond_{(c+y,d)}$.

5.2.1 EXPSPACE-hardness results

We start giving the hardness results presented in the following Lemma.

Lemma 20 *For $\text{PMITL}_{0,\infty}$ formulas and L/U automata, the following problems are* EXPSPACE*-hard:*

1. *Deciding the universality of the S-sets (resp. V-sets) for $\text{PMITL}_{\Diamond}$ formulas.*
2. *Deciding the emptiness of S-sets (resp. V-sets) for $\text{PMITL}_{\Box}$ formulas.*
3. *Both deciding emptiness and deciding the universality of S-sets (resp. V-sets) for formulas whose parameterized operators are either of the form $\Box_{(c+y,d)}$ or of the form $\Diamond_{(c+x,d)}$.*

Proof We only give the proofs for the set $S(\varphi)$. The proofs for $S(\mathcal{A}, \varphi)$ can be obtained reducing the corresponding results for $S(\varphi)$ by considering the automaton $\mathcal{A}$ accepting all the timed sequences. Concerning to the V-sets, we can reduce the dual problems for the S-sets by Lemma 10 and Proposition 9. For our proofs, we reduce the satisfiability problem of fragments of MITL where the intervals of the form $(c, c+1)$ are allowed either only on the operator $\Diamond$ or only on the operator $\Box$, and the rest of the intervals used in the subscripts of the temporal operators have one of the endpoints which is either 0 or ∞. The satisfiability problem in each such fragment is known to be EXPSPACE-complete (see [Alur et al., 1996]).

Consider any MITL formula φ from the above fragment where $(c, c+1)$ is allowed as subscript only of the operator $\Diamond$. Fix a parameter $x \in U$, and rewrite φ to a formula φ' where each operator of the form $\Diamond_{(c,c+1)}$ is replaced with $\Diamond_{(c,c+1+x)}$ and any other part of the formula stays unchanged. We claim that φ is satisfiable if and only if $S(\varphi')$ is universal. To see this first observe that all the parameter valuations are admissible, and therefore, "$S(\varphi')$ is universal" means that $S(\varphi') = \mathbb{N}$. Thus, if $S(\varphi')$ is universal

then $0 \in S(\varphi')$, and hence φ is satisfiable. Vice-versa, if φ is satisfiable, by Lemma 6 we get that $S(\varphi')$ is universal. Therefore, from the above result on the MITL fragments, we get that checking the universality of $S(\varphi)$ is EXPSPACE-hard.

To show EXPSPACE-hardness of the emptiness problem of $S(\varphi)$ when only parameterized operators of the form $\Box_{(c,d+y)}$ are allowed, we reason similarly. We consider now any MITL formula φ from the above fragment where $(c, c+1)$ is allowed as subscript only of the operator $\Box$. Fix a parameter $y \in L$, and rewrite φ to a formula φ' where each operator of the form $\Box_{(c,c+1)}$ is replaced with $\Box_{(c,c+1+y)}$ and any other part of the formula stays unchanged. Thus, if φ is satisfiable then trivially $S(\varphi')$ is not empty (it contains at least 0). Viceversa, if $S(\varphi')$ is not empty, then 0 must belong to $S(\varphi')$, by Lemma 6, and therefore φ is satisfiable. Thus, the claimed result follows from the complexity of the considered MITL fragment.

For the fragments of $\text{PMITL}_{0,\infty}$ where the only parameterized temporal operators are either of the form$\Diamond_{(c+y,d)}$, or of the form $\Box_{(c+x,d)}$, again we reduce the satisfiability problem for the MITL fragments considered above. In particular, we rewrite each operator of the form $\Diamond_{(c,c+1)}$ with $\Diamond_{(c+y,c+1)}$ and any other part of the formula stays unchanged. In this case, we observe that the only admissible value for y is 0, thus testing the emptiness of $S(\varphi')$ coincides with testing its universality and φ is satisfiable if and only if $S(\varphi')$ is not empty. Therefore, the claimed result again follows from the EXPSPACE-hardness of satisfiability for the considered fragment of MITL. The case of the operator $\Box_{(c+x,d)}$, for $x \in U$, is analogous, and thus we omit further details. $\Box$

From the above Lemma 20 and Theorem 19, we can state the following theorem.

Theorem 21 *The problems of checking the emptiness and the universality of each of the S-sets and V-sets for* $\text{PMITL}_{0,\infty}$ *formulas and* L/U *automata are* EXPSPACE*-complete.*

The hardness results from Lemma 20 leave open the computational complexity for some of the considered decision problems in the fragments $\text{PMITL}_{\Diamond}$ and $\text{PMITL}_{\Box}$.

In the rest of this chapter, we show that such decision problems are indeed PSPACE-complete. This is an interesting result, since these fragments include $\text{P}_{0,\infty}\text{MITL}_{0,\infty}$ and capture meaningful properties (see the example from Section 4.5).

5.2.2 PSPACE-hardness results

For a sequence α, we denote with $S_\alpha(\varphi)$ the set of parameter valuations v such that the sequence α satisfies φ under valuation v. Observe that $S(\varphi) = \bigcup_\alpha S_\alpha(\varphi)$ holds.

We start showing PSPACE membership for the emptiness problems in PMITL$_\Diamond$. The next lemma is the crucial result for reducing such problems for the S-sets to the same problems in P$_{0,\infty}$MITL$_{0,\infty}$.

Lemma 22 *Let φ be a well defined PMITL$_\Diamond$ formula containing a subformula χ of the form $\Diamond_I \psi$, for $I = (c, d + z)$. Denote with φ' the formula obtained from φ by replacing χ with the subformula $\chi' = \Box_{]0,c]} \Diamond_{I-c} \psi$. For each timed sequence α, the following properties hold:*

a) $S_\alpha(\varphi') \subseteq S_\alpha(\varphi)$.

b) $S_\alpha(\varphi) = \varnothing \iff S_\alpha(\varphi') = \varnothing$.

Proof Consider first part (a). If $S_\alpha(\varphi') = \varnothing$ then the assertion is trivially true. Otherwise, assume by contradiction that φ is a minimal formula containing χ such that there is a $v \in S_\alpha(\varphi') \setminus S_\alpha(\varphi)$. The proof proceeds by case inspection on the possible forms of φ.

Consider the case $\varphi = \chi$. Let us consider only the case that I is the closed interval $[c, d + z]$, the other cases being analogous. Since $v \in S_\alpha(\varphi')$, for all $t \in]0, c]$, the subformula ψ must be true for some $t' \in [t, t + d + v(z) - c]$, and in particular, ψ must be true in the interval $[c, d + v(z)]$. This suffices to conclude that $v \in S_\alpha(\varphi)$, thus contradicting the hypothesis.

The remaining cases are quite simple and we consider only some sample cases. In particular, if φ is of the form $\varphi_1 \wedge \varphi_2$, and χ is a subformula of φ_1, clearly $S_\alpha(\varphi') = S_\alpha(\varphi'_1) \cap S_\alpha(\varphi_2)$ and by hypothesis $S_\alpha(\varphi'_1) \subseteq S_\alpha(\varphi_1)$, where φ'_1 denotes the formula obtained from φ_1 by replacing χ with χ'. Therefore, $S_\alpha(\varphi'_1) \cap S_\alpha(\varphi_2) \subseteq S_\alpha(\varphi_1) \cap S_\alpha(\varphi_2) = S_\alpha(\varphi)$, and thus we contradict the hypothesis. The disjunction of two subformulas is analogous.

If the main operator of φ is a temporal operator, the key observation in proving the corresponding case is the following. If a subformula γ' of φ' holds at time t of a timed sequence α, denoting with α_t the sequence that matches the suffix of α from t, then by

hypothesis $S_{\alpha_t}(\gamma') \subseteq S_{\alpha_t}(\gamma)$ and thus we get $S_\alpha(\varphi') \subseteq S_\alpha(\varphi)$. As a sample case for this class of formulas, consider φ of the form $\varphi_1 \,\mathcal{U}_I\, \varphi_2$ and let φ_1' denote the formula obtained from φ_1 by replacing χ with χ'. From $v \in S_\alpha(\varphi')$, we get that $(\alpha, v) \models \varphi'$. From the semantics this means that there is a time $t \in I_v$ such that $(\alpha, v, t) \models \varphi_2$ and $(\alpha, v, t') \models \varphi_1'$ for all $0 < t' < t$. By hypothesis, $S_\beta(\varphi_1') \subseteq S_\beta(\varphi_1)$ for each β, thus also $S_{\alpha_{t'}}(\varphi_1') \subseteq S_{\alpha_{t'}}(\varphi_1)$. Therefore, $(\alpha, v, t') \models \varphi_1$ for all $0 < t' < t$, and thus $(\alpha, v) \models \varphi$ contradicting the hypothesis.

To prove part (b) of (1), observe that directly from part (a) of (1), we get that $S_\alpha(\varphi) = \varnothing$ implies $S_\alpha(\varphi') = \varnothing$. For the other direction, we show a stronger result. If $v \in S(\varphi)$ then $v' \in S(\varphi')$ where v' is defined as $v'(z) = v(z) + c$ and $v'(y) = v(y)$ for $y \neq z$. We can prove this result by structural induction on the formulas φ which contain χ as a subformula.

The base case is $\varphi = \chi$. From $v \in S(\varphi)$, we get that the subformula ψ is true along α starting at some time $t' \in [c, d + v(z)]$. Clearly, for all $0 < t \leq c$, we have that $t \leq t'$ and $t' \leq d + v(z)$. Thus, $t \leq t' \leq d + v'(z) - c$. From $t > 0$, we get that $t \leq t' \leq d + v'(z) - c + t$. Recall that $I - c = [0, d - c + z]$. Therefore, $\Diamond_{[0,d-c+z]}\psi$ holds at all $t \in]0, c]$. Thus $(\alpha, v') \models \varphi'$.

For the induction step, the cases involving Boolean operators are quite straightforward from the semantics of the operator as in the proof of part (a) and thus we omit further details on them. For the cases involving the temporal operators the key observation is to exploit the polarity of the temporal operators using z (which is in U). Therefore, for each subformula γ, by Lemma 6, we get that $v' \in S_\beta(\gamma)$ whenever $v \in S_\beta(\gamma)$. The rest of the proof simply uses the semantics of the operators and the induction hypothesis, thus we omit further details. □

Fix a formula φ of PMITL$_\Diamond$. By applying transformations as in the above Lemma 22 to φ, from inside out, we can generate a sequence of formulas $\varphi = \varphi_1, \ldots, \varphi_n = \varphi'$ where the last one is a P$_{0,\infty}$MITL$_{0,\infty}$ formula. Since at each step of the transformation we can apply the above lemma, we get that the properties (a) and (b) stated in the lemma also holds for this φ and φ. Moreover, observe that each transformation only adds one subformula, therefore φ' has size linear in the size of φ.

From the property (b), it is correct to check the emptiness of $S(\varphi)$ by checking the emptiness of $S(\varphi')$, and thus the described transformations give a reduction of

such problem for PMITL$_\Diamond$ formulas to the same problem in P$_{0,\infty}$MITL$_{0,\infty}$. Since this last is known to be in PSPACE, we get PSPACE membership also for the problem of checking the emptiness of the $S(\varphi)$ sets in PMITL$_\Diamond$. We can repeat the same arguments for showing PSPACE-membership of checking the emptiness of the $S(\mathcal{A}, \varphi)$ sets for a given PTA $\mathcal{A}$. Moreover, by Proposition 9 and Lemma 10, we get also PSPACE-membership of the universality problems for the V-sets in PMITL$_\Box$. Therefore the following theorem holds.

Theorem 23 *Given a formula φ in PMITL$_\Diamond$, and an L/U automaton $\mathcal{A}$, then checking the emptiness of the sets $S(\varphi)$ and $S(\mathcal{A}, \varphi)$ is* PSPACE*-complete.*

Given a formula φ in PMITL$_\Box$, and an L/U automaton $\mathcal{A}$, then checking the universality of the sets $V(\varphi)$ and $V(\mathcal{A}, \varphi)$ is PSPACE*-complete.*

Now, we turn our attention to the universality problems for the S-sets in PMITL$_\Box$. The next lemma is crucial to prove membership to PSPACE for such problems. We omit the proof here, since it is very similar to the proof of Lemma 22, and in fact what we prove here is essentially dual to what is shown there. Observe that part (b) of the following lemma implies that "$S_\alpha(\varphi)$ is universal if and only if $S_\alpha(\varphi')$", however this last cannot be used to derive an algorithm to show universality of the S-sets of PMITL$_\Box$. Indeed, the part (b) of the following lemma is dual to the assertion used to prove part (b) of Lemma 22.

Lemma 24 *Let φ be a well defined PMITL$_\Box$ formula containing a subformula χ of the form $\Box_I \psi$, for $I = (c, d+z)$. Denote with φ' the formula obtained from φ by replacing χ with the subformula $\chi' = \Diamond_{]0,c]} \Box_{I-c} \psi$. For each timed sequence α, the following properties hold:*

a) $S_\alpha(\varphi) \subseteq S_\alpha(\varphi')$.

b) For each constant $c' \geq c$ and for each parameter valuation v' such that $v' \in S_\alpha(\varphi')$ and $v'(z) > c'$, denoting with v the parameter valuation defined as $v(z) = v'(z) - c'$ and $v(y) = v'(y)$ for each $y \neq z$: $v \in S_\alpha(\varphi)$ holds.

Fix a formula φ of PMITL$_\Box$. By applying transformations as in the above Lemma 24 to φ, from inside out, we can generate a sequence of formulas $\varphi = \varphi_1, \ldots, \varphi_n = \varphi'$ where the last one is a P$_{0,\infty}$MITL$_{0,\infty}$ formula. Since at each step of the transformation

we can apply the above lemma, we get that property (a) of the lemma also holds for this φ and φ. By part (b) of the lemma and Lemma 6 (the parameters used in the temporal operators involved in the transformation are all from L), we get the following:

Lemma 25 *There exists a constant c_{max} such that for each parameter valuation $v' \in S_\alpha(\varphi')$ with $v'(y) > c_{max}$ for all parameters $y \in L$, given a parameter $z \in L$ and denoting with v the parameter valuation defined as $v(z) = v'(z) - c_{max}$ and $v(y) = v'(y)$ for each $y \neq z$: $v \in S_\alpha(\varphi)$ holds.*

(Note that we can choose as c_{max} the maximum over the constants c used in the applications of part (b) of Lemma 24 to obtain φ' from φ.)

For the above formulas φ and φ'. We can show the following:

Claim 1 *$S(\varphi)$ is universal if and only if $S(\varphi')$ is universal.*

Proof Since the domain of the admissible valuations for φ and φ' is the same (i.e., $\mathcal{D}(\varphi) = \mathcal{D}(\varphi')$), from part (a) of Lemma 24 we trivially get the "only if" direction. For the converse direction. Suppose that there is a $v \notin S(\varphi)$. Let c_{max} be as in Lemma 25 and define $v'(z) = v(z) + c_{max}$ for all parameters z. Now, let $L = \{z_1, \ldots, z_n\}$. Define a sequence of parameter valuations $v_0, \ldots, v_n$ such that $v_0 = v'$, $v_n = v$ and for $i = 1, \ldots, n$, v_i is defined such that $v_i(z_i) = v_{i-1}(z_i) - c_{max}$ and $v_i(y) = v_{i-1}(y)$ for all $y \neq z_i$. By Lemma 25, we have that for a timed sequence α, if $v' \in S_\alpha(\varphi')$ then also $v \in S_\alpha(\varphi)$. Therefore, if $v' \in S(\varphi')$ then also $v \in S(\varphi)$, which proves the claim. □

Finally, observe that each transformation only adds one subformula, therefore φ' has size linear in the size of φ. From the above Claim, it is correct to check the universality of $S(\varphi)$ by checking the universality of $S(\varphi')$, and thus the described transformations give a reduction of such problem for PMITL$_\Box$ formulas to the same problem in P$_{0,\infty}$MITL$_{0,\infty}$. Since this last is known to be in PSPACE, we get PSPACE membership also for the problem of checking the universality of the $S(\varphi)$ sets in PMITL$_\Box$. We can repeat the same arguments for showing PSPACE-membership of checking the universality of the $S(\mathcal{A}, \varphi)$ sets for a given PTA $\mathcal{A}$. Moreover, by Proposition 9 and Lemma 10, we get also PSPACE-membership of the emptiness problems for the V-sets in PMITL$_\Diamond$. Therefore the following theorem holds.

Theorem 26 *Given a formula φ in PMITL$_{\Diamond}$, and an L/U automaton $\mathcal{A}$, then checking the emptiness of the sets $V(\varphi)$ and $V(\mathcal{A}, \varphi)$ is* PSPACE*-complete.*

Given a formula φ in PMITL$_{\Box}$, and an L/U automaton $\mathcal{A}$, then checking the universality of the sets $S(\varphi)$ and $S(\mathcal{A}, \varphi)$ is PSPACE*-complete.*

In Table 5.1 we summarize the computational complexity of the considered decision problems for the logic PMITL and its fragments.

Logic	**Emptiness**	**Universality**
PMITL	EXPSPACE-complete	EXPSPACE-complete
PMITL$_{0,\infty}$	EXPSPACE-complete	EXPSPACE-complete
PMITL$_{\Diamond}$	PSPACE-complete	EXPSPACE-complete
PMITL$_{\Box}$	EXPSPACE-complete	PSPACE-complete
P$_{0,\infty}$MITL$_{0,\infty}$	PSPACE-complete	PSPACE-complete

Table 5.1: *Summary of the computational complexity of the emptiness and universality problems for the sets $S(\varphi)$, $S(\mathcal{A}, \varphi)$, $V(\varphi)$ and $V(\mathcal{A}, \varphi)$ in the studied syntactic fragments of PMITL.*

5.3 Decidable extensions

In this section, we discuss two generalizations of the presented results regarding PMITL. In the first one, we extend our logic by allowing parametric expressions which are *linear expressions* of the parameters. The other generalization is a more general formulation of the considered decision problems. We consider decision problems over the S-sets and the V sets expressed as queries where each parameter is quantified either existentially or universally. In such formulation, the emptiness problem corresponds to a query where all the parameters are quantified existentially and the universality problem corresponds to one where all the parameters are quantified universally.

5.3.1 PMITL$_E$: syntax and decidability results

We discuss a syntactic extension of PMITL, that we call PMITL$_E$, showing that the considered decision problems are EXPSPACE-complete as for PMITL. We introduce

first some notation.

A *linear expression* e is an expression of the form $c_0 + c_1x_1 + \ldots + c_mx_m$ with $c_0, c_1, \ldots, c_m \in \mathbb{Z}$ and $x_1, x_2, \ldots, x_m \in L \cup U$. We say that a parameter p_i *occurs positively* in e if $c_i > 0$ and *occurs negatively* in e if $c_i < 0$. Given a parameter valuation v and a *linear expression* e , with $v(e)$ we denote the integer $c_0 + c_1v(x_1) + \ldots + c_mv(x_m)$, obtained by evaluating the parameters that occurs in e by v. For an interval $I = (e, e')$, we denote with I_v the interval $(v(e), v(e'))$, that is the interval I where the expressions are replaced with their valuations. We denote with e_u (resp.e_l) a linear expression over parameters $U \cup L$ such that each parameter from U (resp.L) occurs positively and each parameter from L (resp. U) occurs negatively.

We now introduce the logic PMITL$_E$. The syntax of PMITL$_E$ formulas over the set of atomic propositions AP is defined by the following grammar:

$$\varphi \;:=\; p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \,\mathcal{U}_H\, \varphi \mid \varphi \,\mathcal{R}_J\, \varphi$$

where $p \in AP$, and H and J are either defined as for PMITL or as follows:

- for $H = (a, b)$, $a \in \mathbb{N}$ and $b = e_u$;
- for $J = (a, b)$, $a \in \mathbb{N}$ and $b = e_l$.

Note that we have extended to *linear expression* e only the left end-points of the intervals. The reason is that by allowing *linear expressions* also in the left endpoints the decision problems become undecidable (see Section 5.4). The semantics given for PMITL formulas directly applies to PMITL$_E$ formulas (we have only changed the concept of parametric expression and thus all the semantic changes are absorbed by the valuation of linear expressions).

To prove the satisfiability, validity and model-checking problems, we can repeat exactly the construction given in Section 4.7. In fact, in all the results shown in that section, we can postpone handling the parametric linear expressions up to solving the decision problems for the Büchi L/U automaton corresponding to the starting formula (there is no need to valuate the intervals up to that point, and thus the parametric linear expressions are carried over through all the transformations). Recall that the parametric linear expressions are already allowed in the time constraints of the L/U automata given by Bozzelli and La Torre [2009], and that the recalled Theorem 5 has been shown for this more general definition of such automata. Therefore, we get the

following theorem.

Theorem 27 *Given a PMITL$_E$ formula φ and an L/U automaton $\mathcal{A}$, checking the emptiness and the universality of the sets $S(\varphi)$ and $S(\mathcal{A}, \varphi)$ is* EXPSPACE*-complete.*

5.3.2 A general decision problem over the set of parameter valuations and complexity results

In Chapter 4, we have addressed the emptiness and universality queries for the sets $S(\varphi)$, $S(\mathcal{A}, \varphi)$, $V(\varphi)$ and $V(\mathcal{A}, \varphi)$. Here we define a general decision problem over such sets that includes the above emptiness and universality problems as instances.

Fix a PMITL formula φ. In the following, we say that an i-tuple of natural numbers $(a_1, \ldots, a_i)$ is *admissible* if there is a valuation $v \in \mathcal{D}(\varphi)$ that assigns a_j to z_j, for all $j \in \{1, \ldots, i\}$.

Let $z_1, \ldots, z_n$ be a given ordering of the parameters used in φ and $Q_1, \ldots, Q_n \in \{\forall, \exists\}$. We inductively define the sets $S_0, \ldots, S_n$ as follows. We let $S_n = S(\varphi)$ and for $i \in \{0, \ldots, n-1\}$: if $Q_{i+1} = \exists$, S_i is the set $\{(a_1, \ldots, a_i) \mid (a_1, \ldots, a_i, a_{i+1}) \in S_{i+1}$ for some a_{i+1} such that $(a_1, \ldots, a_i, a_{i+1})$ is admissible$\}$; otherwise, i.e., $Q_{i+1} = \forall$, S_i is the set $\{(a_1, \ldots, a_i) \mid (a_1, \ldots, a_i, a_{i+1}) \in S_{i+1}$ for all the a_{i+1} such that $(a_1, \ldots, a_i, a_{i+1})$ is admissible$\}$.

In the following we denote the above set S_0 as $S(Q_1 z_1 \ldots Q_n z_n.\varphi)$.

Decision problem The $Q_1 z_1 \ldots . Q_n z_n$ query over $S(\varphi)$ asks whether the set $S(Q_1 z_1 \ldots Q_n z_n.\varphi)$ is not empty. Analogously, we define the $Q_1 z_1 \ldots . Q_n z_n$ queries over the sets $S(\mathcal{A}, \varphi)$, $V(\varphi)$ and $V(\mathcal{A}, \varphi)$.

Observe, that the non emptiness and universality problems considered in the previous sections correspond respectively to fully existential queries (i.e., each Q_i is $\exists$) and fully universal queries (i.e., each Q_i is $\forall$) over such sets.Thus, by Theorem 19 we immediately have the following:

Remark 28 *Let φ be a PMITL formula over the parameters $z_1, \ldots, z_n$, A be an L/U automaton over the same parameters and Γ be one among the sets $S(\varphi)$, $S(\mathcal{A}, \varphi)$, $V(\varphi)$ and $V(\mathcal{A}, \varphi)$. If either $Q_1, \ldots, Q_n \in \{\exists\}$ or $Q_1, \ldots, Q_n \in \{\forall\}$, then deciding a query $Q_1 z_1 \ldots . Q_n z_n$ over Γ is* EXPSPACE*-complete.*

Due to the polarity of the parameterized operators and the fact that the admissible values for each parameter are independent from the value assigned to the other parameters, in the introduced queries, the position of the existential quantifiers coupled with a parameter from L and of the universal quantifiers coupled with a parameter from U is not relevant.

In fact, consider a query $Q_1 z_1 \ldots . Q_n z_n$ over $S(\varphi)$. Fix a parameter z_i and denote with m_i its minimum admissible value. If $Q_i = \exists$ and $z_i \in L$, whenever $(a_1, \ldots, a_i, \ldots, a_n) \in S(\varphi)$, by Lemma 6 we have also $(a_1, \ldots, m_i, \ldots, a_n) \in S(\varphi)$, independently of the choice of a_j for each $j \neq i$. Thus, a query which existentially quantifies z_i is satisfied if and only if it is satisfied by assigning z_i with m_i, and therefore, we can move $Q_i z_i$ to any other position of the sequence $Q_1 z_1 \ldots . Q_n z_n$ without altering its validity.

In the other case, i.e., $Q_i = \forall$ and $z_i \in U$, again by Lemma 6 and analogously to the previous case, we can argue that a query which universally quantifies z_i is satisfied if and only if is satisfied by assigning z_i with m_i. Therefore, we can move $Q_i z_i$ to any other position of the sequence $Q_1 z_1 \ldots . Q_n z_n$ without altering its validity.

The above observations are the main arguments to show the following lemma:

Lemma 29 *Let φ be a PMITL formula over the parameters $z_1, \ldots, z_n$ and $Q_1, \ldots, Q_n \in \{\forall, \exists\}$. Denote with Γ any of the sets $S(\varphi)$, $S(\mathcal{A}, \varphi)$, $V(\varphi)$ and $V(\mathcal{A}, \varphi)$.*

For each query $Q_1 z_1 \ldots . Q_n z_n$ over Γ there is an equivalent query $Q_{i_1} z_{i_1} \ldots . Q_{i_n} z_{i_n}$ over Γ such that:

- *$i_1, \ldots, i_n$ is a permutation of $1, \ldots, n$,*
- *$Q_{i_1}, \ldots, Q_{i_k} \in \{\exists\}$ and $z_{i_1}, \ldots, z_{i_k} \in L$, for some $k \in \{0, \ldots, n\}$,*
- *$Q_{i_{k+1}}, \ldots, Q_{i_{k+j}} \in \{\forall\}$ and $z_{i_{k+1}}, \ldots, z_{i_{k+j}} \in U$, for some $j \in \{0, \ldots, n-k\}$,*
- *for all $h \in \{k+j+1, \ldots, n\}$, $Q_{i_h} = \exists$ if and only if $z_{i_h} \in U$.*

Moreover, the sequence $i_1, \ldots, i_n$ can be effectively computed in linear time.

The above lemma suggests a simple algorithm to decide a query $Q_1 z_1 \ldots . Q_n z_n$ over the sets $S(\varphi)$, $S(\mathcal{A}, \varphi)$, $V(\varphi)$ and $V(\mathcal{A}, \varphi)$ when in the query either all the parameters from U correspond to the quantifier $\forall$, or all the parameters from L correspond to the quantifier $\exists$. Suppose that the ordering of the parameters is as stated in

Lemma 29. The algorithm starts by eliminating the parameters in subformulas of the forms $\Diamond_{(c+z,d)}\psi$ and $\Box_{(c+z,d)}\,\psi$, as in Theorem 19. Then, we assign all the parameters from L, in the scope of the quantifier $\exists$, and all the parameters from U, in the scope of the quantifier $\forall$, with the minimum admissible value in their domains. After this step, the remaining parameters are either all from U and in the scope of the quantifier $\exists$ or all from L and in the scope of the quantifier $\forall$. Thus, we use the decision algorithm as by Remark 28 on the resulting query. Therefore, we get the following theorem.

Theorem 30 *Let φ be a PMITL formula over the parameters $z_1, \ldots, z_n$, $Q_1, \ldots, Q_n \in \{\forall, \exists\}$, $\mathcal{A}$ be an L/U automaton over the same parameters and Γ be one among the sets $S(\varphi)$, $S(\mathcal{A}, \varphi)$, $V(\varphi)$ and $V(\mathcal{A}, \varphi)$. If either one of the following cases holds:*

- *$Q_i = \exists$ for each $z_i \in L$, or*
- *$Q_i = \forall$ for each $z_i \in U$,*

then deciding a query $Q_1 z_1 \ldots . Q_n z_n$ over Γ is EXPSPACE*-complete.*

Observe that the above theorem holds in particular when $z_1, \ldots, z_n$ are all from L or all from U. Also, note that in general the above decision algorithm is not correct for arbitrary queries $Q_1 z_1 \ldots . Q_n z_n$ on arbitrary PMITL formulas and L/U automata. We are not aware of a solution for the general decision problem we have stated, and it is not clear to us whether the problem is even decidable.

5.4 Parameterization of Time Intervals

The need for restricting the use of each parameter with temporal operators of the same polarity has been already addressed by Alur et al. [2001], Hune et al. [2002] and by Bozzelli and La Torre [2009] for parametric temporal logics and parametric timed automata. The arguments used there also apply to PMITL and therefore we omit further discussion on this aspect.

In this section, we focus on the other restrictions we have placed on the definition of parameterized intervals of PMITL and PMITL$_E$. In particular, we relax the restriction that at most one of the end-points of an interval is a parametric expression,

and define three natural ways of adding parameters to both the end-points of the intervals. Unfortunately, none of the proposed parameterizations leads to a decidable logic already for the logic PMITL. Moreover, we show that for PMITL$_E$ also when only one of the interval end-points can be parametric, but we allow parameterized intervals with parameters either in the left or the right end-point, the resulting logic becomes undecidable.

For simplicity, in this section we consider only the satisfiability problem, that is the problem of checking the emptiness of the set $S(\varphi)$. The results for the other considered sets can be achieved similarly.

5.4.1 Parameterized time-shifts of intervals

In the first parameterization, we consider parameterized time-shifts of intervals. More precisely, with $\mathcal{L}_1$ we denote the logic obtained by augmenting MITL with parameterized intervals of the form $(c + x, d + x)$, such that (c, d) is not singular. Observe, that operators with this kind of intervals do not have polarity.

Theorem 31 *The problem of checking the emptiness of $S(\varphi)$ for any φ in $\mathcal{L}_1$ is undecidable. In particular, this holds already for the fragment of $\mathcal{L}_1$ with a single parameter x and where all parametric intervals are of the form $(x, x + 1)$.*

Proof We reduce the problem of checking the emptiness of sets $S(\varphi)$ for a formula φ of the logic PLTL (parametric LTL) augmented with the equality in the parametric constraints coupled with the temporal operators. This problem is known to be undecidable even when a single parameter is allowed [Alur et al., 2001]. The logic PLTL is essentially the logic P$_{0,\infty}$MITL$_{0,\infty}$ augmented with the next-time operator and with a discrete semantics (i.e., given with respect to infinite sequences of nodes labeled with atomic propositions).

The main idea is to capture a discrete-time sequence σ with a timed sequence α such that a position i in σ is captured by the interval $[i, i + 1[$ within α. In this way, we can encode an equality constraint of the form $= x$ with the interval $[x, x + 1[$ and the next-time operator with the operator $\Diamond_{[1,2[}$. Fix a formula φ from PLTL extended with equality and a single parameter x, and such parameter is only coupled with equality in the constraints. We translate φ into a formula φ' of $\mathcal{L}_1$ which uses only one parameter

x. The formula φ' is the conjunction of two formulas. The first formula φ'_1 captures the timed sequences that encode the discrete sequences as described above. The second formula φ'_2 captures the requirements expressed by φ on such timed sequences.

We use a fresh atomic proposition p to denote the change of the discrete time (the "tick" of the discrete time clock). We require that $p \in \alpha(t)$ for all $t \in \mathbb{N}$ and $p \notin \alpha(t)$, otherwise. Thus the formula φ'_1 is:

$$p \wedge \Box_{[0,\infty[}(p \rightarrow (\neg p \,\mathcal{U}_{=1}\, p)) \wedge \bigwedge_{a \in AP} \Box_{[0,\infty[}((p \wedge a) \rightarrow \Box_{[0,1[}\, a),$$

where $\neg p \,\mathcal{U}_{=1}\, p$ is just an abbreviation for $(\neg p \,\mathcal{U}_{\leq 1}\, p) \wedge (\neg p \,\mathcal{U}_{\geq 1}\, p)$, and AP is the set of atomic proposition used in φ.

The formula φ'_2 is obtained from φ through the translation function τ defined inductively as follows, for any subformula ψ:

- if $\psi = p$ or $\psi = \neg p$ for $p \in AP$, then $\tau(\psi) = \psi$;
- for $\circ \in \{\wedge, \vee\}$, if $\psi = \psi_1 \circ \psi_2$, then $\tau(\psi) = \tau(\psi_1) \circ \tau(\psi_2)$;
- if $\psi = \bigcirc \psi'$, then $\tau(\psi) = \Diamond_{[1,2[}\tau(\psi')$;
- for $\nabla \in \{\mathcal{U}, \mathcal{R}\}$, if $\psi = \psi_1 \nabla_{\approx c} \psi_2$, then $\tau(\psi_1) \nabla_{\approx c} \tau(\psi_2)$ (where $c \in \mathbb{N}$ and $\approx \in \{\leq, <, >, \geq\}$);
- for $\nabla \in \{\mathcal{U}, \mathcal{R}\}$, if $\psi = \psi_1 \nabla_{=x} \psi_2$, then $\tau(\psi_1) \nabla_{[x,x+1[} \tau(\psi_2)$.

Denoting $\psi'_2 = \tau(\varphi)$, we have that $S(\varphi)$ is empty if and only if $S(\varphi'_1 \wedge \varphi'_2)$ is empty, and thus the theorem holds. □

5.4.2 Full parameterization of intervals

We extend PMITL with parameterized intervals where both left end-points and right end-points are in $\mathcal{E}(L) \cup \mathcal{E}(U)$. More precisely, given $x \in U$ and $y \in L$, we consider *fully parameterized* intervals which can be of the form $(c + y, d + x)$, when used as subscripts of *until* operators, and of the form $(c + x, d + y)$, when used as subscripts of *release* operators. We denote this logic $\mathcal{L}_2$.

Theorem 32 *The problem of checking the emptiness of $S(\varphi)$ for any φ in $\mathcal{L}_2$ is undecidable.*

Proof Consider the formula $\varphi = \Diamond_{[y,x+1]}\psi \wedge \Box_{[x+1,y+2]} true$. For an admissible parameter valuation v, it holds that $v(y) < v(x) + 1$ and $v(x) + 1 < v(y) + 2$ from which we obtain that $v(x) = v(y)$. Thus, φ is equivalent to the formula $\Diamond_{[z,z+1]}\psi$ of $\mathcal{L}_1$. Therefore, the theorem follows from Theorem 31. □

Another way of obtaining full parametrization of the intervals is to use a parameter for translating the interval in time and the other to adjust the width of the interval. Let $\mathcal{L}_3$ denote the corresponding logic. We can show that this logic is also undecidable by using the following reduction. Given an interval $(c + y, d' + y + x)$, we obtain the interval $(c + y', d + x')$ by the linear transformation: $y' = y$, $x' = c + 1 + x + y' - d$, and $d' = c + 1$. Thus, from Theorem 32 we get:

Theorem 33 *The problem of checking the emptiness of $S(\varphi)$ for any φ in $\mathcal{L}_3$ is undecidable.*

5.4.3 Parameters as left end-points in PMITL$_E$

We have defined PMITL$_E$ such that the parametric linear expressions can be used only as right end-points of the parameterized intervals. Here we relax this restriction and admits such expressions also as left end-points of the intervals, but we keep the restriction that only one endpoint is parameterized. We call $\mathcal{L}_4$ the resulting logic. The following theorem holds.

Theorem 34 *The problem of checking the emptiness of $S(\varphi)$ for any φ in $\mathcal{L}_4$ is undecidable.*

Proof Consider the formula $\varphi = \Box_{[x-y,1]} true \wedge \neg\psi\, \mathcal{U}_{]0,x]}\, \psi \wedge \neg\psi\, \mathcal{U}_{[y,\infty[}\, \psi$. For an admissible parameter valuation v, it holds that $0 \leq v(x) - v(y) < 1$, from which we obtain that $v(x) = v(y)$. Thus, φ is equivalent to the formula $\neg\psi\, \mathcal{U}_{=x}\, \psi$. It is known that augmenting PLTL with such kind of formulas results in an undecidable logic [Alur et al., 2001]. We can use a translation as in the proof of Theorem 31 to reduce the emptiness problem for the sets $S(\varphi)$ in this discrete-time logic to show undecidability of the same problem in $\mathcal{L}_4$. □

Chapter 6

Safraless Complementation for Timed Specification

In this chapter we investigate the applicability of automata constructions that avoid determinization for solving the language inclusion problem. Since Safra's determinization procedure is difficult to implement, we present Safraless decision procedures that have recently been investigated, extending them to timed specifications. In particular, we consider the class of alternating event-clock automata, providing algorithms for solving the universality and language inclusion problems for that class of timed automata without resorting to the Safra construction.

6.1 The Language Inclusion Problem

Temporal logics have been adopted as a powerful tool for specifying and verifying concurrent programs [Manna and Pnueli, 1992a; Pnueli, 1977] and one of the most significant developments in this area is the discovery of algorithmic methods for verifying temporal logic properties of finite-state programs [Clarke et al., 1986; Lichtenstein and Pnueli, 1985; Queille and Sifakis, 1982].

This is significative because many synchronization and communication protocols can be modeled as finite-state programs [Liu, 1989; Rudin, 1987].

Finite-state programs can be modeled by transition systems where each state has a bounded description, and hence can be characterized by a fixed number of Boolean

atomic propositions. This implies that a finite-state program can be viewed as a finite propositional Kripke structure and that its properties can be specified using propositional temporal logic. For that reason, to verify the correctness of the program with respect to a desired behavior, one only has to check that the program, modeled as a finite Kripke structure, is a model of (satisfies) the propositional temporal logic formula that specifies that behavior. Hence the name model checking for the verification methods derived from this viewpoint.

In the automata-theoretic approach to verification, we reduce questions about programs and their specifications to questions about automata [Kupferman and Vardi, 2001]. More specifically, questions such as satisfiability of specifications and correctness of programs with respect to their specifications are reduced to questions such as non-emptiness and language containment [Kurshan, 1994; Vardi and Wolper, 1986, 1994].

Programs and properties are formalized as regular languages of infinite words. Any regular language of infinite words is accepted by a nondeterministic Büchi automaton.

Automata on infinite words are used for specification and verification of nonterminating programs. The idea is simple: when a program is defined with respect to a finite set P of propositions, each of the programs states can be associated with a set of propositions that hold in this state. Then, each of the programs computations induces an infinite word over the alphabet 2^P, and the program itself induces a language of infinite words over this alphabet. This language can be defined by an automaton. Similarly, a specification for a program, which describes all the allowed computations, can be viewed as a language of infinite words over 2^P, and can therefore be defined by an automaton.

In practice, we can define the *Language Inclusion* problem as follows.

If we denote with A the non deterministic Büchi automaton that formalizes the program, and with B the non deterministic Büchi automaton that formalizes the specification, the verification problem (if the program satisfies the specification) corresponds to test if $L(A) \subseteq L(B)$ that is if $L(A) \cap L(\neg B) = \varnothing$.

With this procedure, $\neg B$ is obtained by determinization of B. Nevertheless, currently there is no practical algorithms to solve this language inclusion problem. The usual approach through explicit complementation is difficult, as we will see in Section 6.2.

The automata-theoretic approach separates the logical and the combinatorial aspects of reasoning about programs. The translation of specifications to automata handles the logic and shifts all the combinatorial difficulties to automata-theoretic problems.

Language containment is also useful in the context of abstraction, where a large system is replaced by an abstraction whose language is richer, yet its state space is smaller. Such abstractions are particularly useful in the context of parametric verification, where a parallel composition of an unbounded number of processes is abstracted by a composition of a finite number of them [Kesten and Pnueli, 2000; Kesten et al., 2005], and in the context of inheritance and behavioral conformity in object-oriented analysis and design [Harel and Kupferman, 2002].

Other applications regard to the fact that language equivalence is checked by two language-containment tests. For example, the translators from LTL into automata have reached a remarkable level of sophistication (cf. [Gurumurthy et al., 2002]), and it is useful to check their correctness, which involves a language-equivalence test.

6.2 The Complementation Problem for Automata on Infinite Words

The *Complementation problem* for nondeterministic automata on infinite words [Kupferman and Vardi, 2005a] has numerous applications in formal verification. As we have already seen in Section 6.1, in order to check that the language of an automaton A is contained in the language of a second automaton B, one checks that the intersection of A with an automaton that complements B is empty. Many problems in verification and design are reduced to language containment. In model checking, the automaton A corresponds to the system, and the automaton B corresponds to the specification [Kurshan, 1994; Vardi and Wolper, 1994].

While it is easy to complement specifications given in terms of formulas in temporal logic, complementation of specifications given in terms of automata is so problematic, that in practice the user is required to describe the specification in terms of a deterministic automaton (it is easy to complement a deterministic automaton [Aggarwal and Kurshan, 1984; Hardin et al., 1996], or to supply the automaton for the

negation of the specification [Holzmann, 1997].

Efforts to develop simple complementation constructions for nondeterministic automata started early in the 60s, motivated by decision problems of second order logics. Büchi [1962] suggested a complementation construction for nondeterministic Büchi automata that involved a complicated combinatorial argument and a doubly-exponential blow-up in the state space.

Thus, complementing an automaton with n states resulted in an automaton with $2^{2^{O(n)}}$ states. In 1988, Safra introduced an optimal determinization construction, which also enabled a $2^{O(n \log n)}$ complementation construction [Safra, 1988], matching the known lower bound [Michel, 1988].

Another $2^{O(n \log n)}$ construction was suggested by Klarlund [1991], which circumvented the need for determinization. The optimal constructions in [Safra, 1988] and in [Klarlund, 1991] found theoretical applications in the establishment of decision procedures (cf. [Emerson and Jutla, 1991]), but the intricacy of the constructions makes their implementation difficult. We do not know any implementation of Klarlund's algorithm, and the implementation of Safras algorithm [Tasiran et al., 1995] has to cope with the rather involved structure of the states in the complementary automaton.

Kupferman and Vardi [2001] described a simple, optimal complementation of nondeterministic Büchi automata, based on the analysis of runs of universal co-Büchi automata. A report on an implementation of this construction can be found in [Gurumurthy et al., 2003].

The construction was extended to nondeterministic generalized Büchi automata by Kupferman and Vardi [2004].

Beyond its simplicity, the construction has other attractive properties: it can be implemented symbolically [Kupferman and Vardi, 2001], it is amenable to optimizations [Gurumurthy et al., 2003], improvements [Friedgut et al., 2006], and it naturally generates certificates to the verification task [Kupferman and Vardi, 2004].

6.3 Safra's Determinization

The construction of a deterministic ω-automaton equivalent to a non deterministic ω-automaton is an important and recurrent element in decision procedures for various logics [Safra, 1988]. However, when concerned with complexity, due to the prohibitive

cost associated with the determinization process, none of the decision procedures used the determinization process in its entirety.

Emerson and Sistla [1984] developed a special determinization process which is only singly exponential, based on the special properties of the automata associated with linear temporal logic. Vardi and Stockmeyer [1985] reduced the satisfiability problem for various modal logics to the emptiness problem for hybrid tree automata, in order to avoid the natural reduction to emptiness of the Streett [1982] tree automata, which involves determinization.

Given a specification expressed by a deterministic ω-automaton, Vardi and Stockmeyer [1985] developed a polynomial time verification procedure for probabilistic programs. The natural procedure, given a specification by a nondeterministic automaton, is first to determinize the automaton, and then apply the verification procedure. He showed that there is no need for complete determinization in order to apply the procedure, the automaton need only be deterministic in the limit.

The complementation problem is another important problem that arises when using ω-automaton for specification and decision procedures: given an automaton, we have to construct another automaton that accepts the complementary language. Büchi [1962] proved that his automata are closed under complementation , but his proof was not completely constructive.

In the literature, several explicit constructions were given [Büchi, 1973; Choueka, 1974; McNaughton, 1966; Siefkesi, 1970], but all of these involve a doubly exponential blow-up.

6.3.1 Use and disadvantages

Safra [1988] has provided a new determinization construction.

The advantages of this construction are that it is simpler to understand and yields a single exponent upper bound for the general case. More precisely, given a Büchi automaton of size n, the resulting Rabin automaton has $2^{O(n \log n)}$ states and n pairs. This construction is essentially optimal.

Using the small number of accepting pairs in the determinized automaton and a simple conversion from the complement of a deterministic Rabin automaton to a Büchi automaton, which is exponential only in the number of accepting pairs, Safra gives an

alternative simple complementation construction, which improves the known upper bound of Sistla et al. [1987].

For a Büchi automaton of size n, he constructs a complementary Büchi automaton of size $2^{O(n \log n)}$. From the result of Michel [1988] it follows that this construction is essentially optimal.

The determinization construction, showed by Safra, generalizes the subset construction [M.O.Rabin and D.Scott, 1959].

Safra shows a subset tree construction which consists of a tree, where each node maintains the incomplete variant of the subset construction, so the states in the constructed deterministic automaton are ordered trees of subsets of states; this construction is sound and complete.

6.4 Safraless Decision Procedures

Recent research efforts have investigated alternative decision procedures that avoid the use of Safra's construction. We focus on the Progress measure construction, showed by Klarlund [1991] and on the Rank construction, showed by Kupferman and Vardi [2001].

6.4.1 Progress measure construction

The work of Klarlund [1991] presents a novel technique based on progress measures [Klarlund, 1990, 1991; Klarlund and Kozen, 1991] for the complementation of ω-automata. Instead of using usual combinatorial or algebraic properties of transition relations, the author shows that a graph-theoretic approach based on the notion of progress measures is a potent tool for complementing ω-automata.

Using this technique, Klarlund obtains an elementary proof of the classic result that the class of languages defined by Büchi automata is closed under complementation; the complementation result is obtainable in an elementary fashion without the need for a sophisticated determinization construction and without relying on Ramsey's Lemma or other advanced combinatorial tools used in the past.

6.4.2 Rank construction

Different types of automata induce different levels of expressive power, of succinctness, and of complexity. For example, alternating automata [Chandra et al., 1981] have both existential and universal branching modes and are particularly suitable for specification of programs. Even though alternating Büchi automata are as expressive as nondeterministic Büchi automata (both recognize exactly all ω-regular languages), alternation makes Büchi automata exponentially more succinct. That is, translating an alternating Büchi automaton to a nondeterministic one might involve an exponential blow-up [Drusinsky and Harel, 1994].

Since the combinatorial structure of alternating automata is rich, translating specifications to alternating automata is much simpler than translating them to nondeterministic automata. Alternating automata enable a complete partition between the logical and the combinatorial aspects of reasoning about programs, and they give rise to cleaner and simpler verification algorithms [Moller and Birtwistle, 1996; Vardi, 1995].

The ability of alternating automata to switch between existential and universal branching modes also makes their complementation very easy. For example, in order to complement an alternating Muller automaton on infinite words, one only has to dualize its transition function and acceptance condition [Lindsay, 1988; Miyano and Hayashi, 1984].

In contrast, complementation is a very challenging problem for nondeterministic automata on infinite words. In particular, complementing a nondeterministic Büchi automaton involves an exponential blow-up [Michel, 1988; Safra, 1988].

Muller et al. [1986] introduced *weak alternating automata*. In a weak alternating automaton, the automatons set of states is partitioned into partially ordered sets. Each set is classified as accepting or rejecting. The transition function is restricted so that in each transition the automaton either stays at the same set or moves to a set smaller in the partial order. Thus, each run of a weak alternating automaton eventually gets trapped in some set in the partition. Acceptance is then determined according to the classification of this set. The special structure of weak alternating automata is reflected in their attractive computational properties and makes them very appealing. For example, while the best known complexity for solving the membership problem for Büchi alternating automata is quadratic time, we know how to solve the membership problem

for weak alternating automata in linear time [Kupferman et al., 2000].

Weak alternating automata are a special case of Büchi alternating automata. Indeed, the condition of getting trapped in an accepting set can be replaced by a condition of visiting states of accepting sets infinitely often. The other direction, as it is easy to see, is not true. In fact, it is proven in [Muller et al., 1986; Rabin, 1970], that, when defined on trees, a language L can be recognized by a weak alternating automaton iff both L and its complement can be recognized by Büchi nondeterministic automata.

Nevertheless, when defined on words, weak alternating automata are not less expressive than Büchi alternating automata, and they can recognize all the ω-regular languages. To prove this, Muller et al. [1986] and Lindsay [1988] suggest a linear translation of deterministic Muller automata to weak alternating automata. Using, however, the constructions of Muller et al. [1986] and Lindsay [1988] in order to translate a nondeterministic Büchi or co-Büchi automaton A into a weak alternating automaton, one has no choice but to first translate A into a deterministic Muller automaton. Such a determinization involves an exponential blow-up [Safra, 1988].

Even worse, if A is an alternating automaton, then its determinization involves a doubly-exponential blow-up, and hence, so does the translation to weak alternating automata.

To circumvent the need for determinization, Kupferman and Vardi provide a simple complementation algorithm for nondeterministic Büchi automata, describing a quadratic translation of Büchi and co-Büchi alternating automata to weak alternating automata [Kupferman and Vardi, 2001].

The closure of nondeterministic Büchi automata under complementation plays a crucial role in solving decision problems of second-order logics. As a result, many efforts have been put in proving this closure and developing simple complementation algorithms. Büchi [1962] suggested a complementation construction, which indeed solved the problem, yet involved a complicated combinatorial argument and a doubly-exponential blow-up in the state space. Thus, complementing an automaton with n states resulted in an automaton with $2^{2^{O(n)}}$ states. Sistla et al. [1987] suggested an improved construction, with only $2^{O(n^2)}$ states, which is still, however, not optimal.

As we have seen in Section 6.3, Safra [1988] introduced an optimal determinization construction, which also enabled a $2^{O(nlogn)}$ complementation construction, matching the known lower bound [Michel, 1988].

As we have discussed in Section 6.4.1, Klarlund suggested another $2^{O(nlogn)}$ construction [Klarlund, 1991], which circumvented the need for determinization .

While being the heart of many complexity results in verification, the optimal constructions in Safra and Klarlund are complicated. In particular, the intricacy of the algorithms makes their implementation difficult. It is not well know an implementation of Klarlunds algorithm, and the implementation of Safras algorithm [Tasiran et al., 1995] has to cope with the involved structure of the states in the complementary automaton.

The problem is that the lack of a simple implementation is not due to a lack of need. Recall, that in the automata-theoretic approach to verification, we check correctness of a program with respect to a specification by checking containment of the language of the program in a language of an automaton that accepts exactly all computations that satisfy the specification. In order to check the latter, we check that the intersection of the program with an automaton that complements the specification automaton is empty.

As a result, due to the lack of a simple complementation construction, verification tools have to restrict the specification automaton or improvise other solutions. For example, in the verification tool COSPAN [Kurshan, 1994], the specification automaton must be deterministic, in fact it is easy to complement deterministic automata [Clarke et al., 1993].

In the verification tool SPIN [Holzmann, 1991], the user has to complement the automaton by himself; thus, together with the program, SPIN gets as input a nondeterministic Büchi automaton which accepts exactly all computations that do not satisfy the specification.

The complementary automaton constructed in the procedure of Kupferman and Vardi [2001] is similar to the one constructed by Klarlund [1991], but as their construction involves alternation, it is simpler and easily implementable. If we consider a nondeterministic Büchi automaton A, it is possible to complement A by regarding it as a universal co-Büchi automaton. Then, we can use the construction of Kupferman and Vardi, translating this complementary automaton to a weak alternating automaton W. By Miyano and Hayashi [1984], weak alternating automata can be translated to nondeterministic Büchi automata. Applying their exponential, yet simple, translation to W, we end up with a nondeterministic Büchi automaton $\neg A$ that complements A. For

A with n states, the size of $\neg A$ is $2^{O(nlogn)}$, meeting the known lower bound [Michel, 1988].

More specifically, Kupferman and Vardi [2001] show that if a co-Büchi alternating automaton has an accepting run on a word w, then it also has a very structured accepting run on w. After that, the authors employ this structured run in order to translate Büchi and co-Büchi alternating automata to weak alternating automata.

Löding and Thomas [2000] use the structured runs in order to define runs of weak alternating automata as DAGs of bounded width and this enables them to prove the appropriate determinacy result directly. Piterman [2000] uses the structured runs in order to extend linear temporal logic with alternating word automata. The ranks defined by Kupferman and Vardi are closely related to the progress-measures introduced by Klarlund [1990].

Progress measures are a generic concept for quantifying how each step of a program contributes to bringing a computation closer to its specification. Progress measures are also used by Klarlund [1991] for reasoning about automata on infinite words. When Kupferman and Vardi use these ranks, they consider, unlike Klarlund, alternating automata. Consequently, they do not need to follow a subset construction and to consider several ranks simultaneously. Thus, much of the complication introduced by Klarlund is handled by the rich structure of the automata.

6.5 Extension to Timed Specifications

This section presents extensions of Safraless algorithms proposed in the literature for automata on infinite untimed words to the case of automata on infinite timed words. More precisely, we introduce Safraless procedures for computing the complement of an alternating event-clock automaton with Büchi acceptance condition [Di Giampaolo et al., 2010a]. We show that the techniques of Kupferman and Vardi [2001] can be adapted to alternating event-clock automata. That is, given an alternating event-clock automaton with co-Büchi acceptance condition A, we show how to construct, in quadratic time, an alternating event-clock automaton with Büchi acceptance condition B that accepts the same language as A. From that alternating event-clock automaton B, we show how to construct in exponential time a nondeterministic event-clock automaton C with Büchi acceptance condition such that accepts the same language as B

and A. This is done by adapting a classical construction due to Miyano and Hayashi [1984] originally proposed for Büchi automata on infinite (untimed) words. Those procedures then can be used to complement nondeterministic event-clock automata with Büchi acceptance conditions, this in turn leads to algorithms for solving the universality and language inclusion problems for that class of timed automata without resorting to the Safra construction.

6.5.1 Preliminaries

We introduce some notations that will be used in the definition of the language inclusion problem for automata on infinite words. We have already defined in Section 2.1.2 words and timed words, and we are going to define event clocks and alternating event clock automata.

First, we have to point out the importance we give to time divergence, as follows.

Remark 35 (Time divergence) *In the sequel, we formalize the results for languages of timed words that are not necessarily time divergent. Nevertheless, we systematically explain informally how to obtain the results for diverging timed words.*

Event clocks A *clock* is a real-valued variable whose value evolves with time elapsing. We associate, to every letter $\sigma \in \Sigma$, a *history* clock $\overleftarrow{x_\sigma}$ and a *prophecy* clock $\overrightarrow{x_\sigma}$. We denote respectively by $\mathbb{H}_\Sigma$ the set $\{\overleftarrow{x_\sigma} \mid \sigma \in \Sigma\}$ of *history clocks* and by $\mathbb{P}_\Sigma$ the set $\{\overrightarrow{x_\sigma} \mid \sigma \in \Sigma\}$ of *prophecy clocks* on Σ, and we let $\mathbb{C}_\Sigma = \mathbb{H}_\Sigma \cup \mathbb{P}_\Sigma$ be the set of *event-clocks* on Σ. A valuation v of a set of clocks $C \subseteq \mathbb{C}_\Sigma$ is a function $C \to \mathbb{R}^{\geq 0} \cup \{\bot\}$. We denote by $\mathcal{V}(C)$ the set of all valuations of the set of clocks C. We associate to each position $i \geq 0$ of a timed word $\theta = (w, \tau) \in \mathsf{T}\Sigma^\omega \cup \mathsf{T}\Sigma^*$ a unique valuation $\mathsf{Val}_\mathsf{i}^\theta$ of the clocks in $\mathbb{C}_\Sigma$, defined as follows. For any $x \in \mathbb{H}_\Sigma$, $\mathsf{Val}_\mathsf{i}^\theta(x) = \bot$ if there is no $j < i$ s.t. $w_j = \sigma$. Otherwise, $\mathsf{Val}_\mathsf{i}^\theta(x) = \tau_i - \tau_j$ where j is the largest position s.t. $j < i$ and $w_j = \sigma$. Symmetrically, for any $x \in \mathbb{P}_\Sigma$, $\mathsf{Val}_\mathsf{i}^\theta(x) = \bot$ if there is no $j > i$ s.t. $w_j = \sigma$. Otherwise, $\mathsf{Val}_\mathsf{i}^\theta(x) = \tau_j - \tau_i$ where j is the least position s.t. $j > i$ and $w_j = \sigma$. Intuitively, this means that, when reading the timed word θ, the history clock $\overleftarrow{x_\sigma}$ always records the amount of time elapsed since the last occurrence of σ, and the prophecy clock $\overrightarrow{x_\sigma}$ always tells us the amount of time before the next occurrence of σ. For a valuation $v \in \mathcal{V}(C)$ such that $\forall x \in \mathbb{P}_\Sigma \cap C\colon\ v(x) \geq d$, we denote by $v + d$

the valuation from $\mathcal{V}(C)$ that respects the following two conditions. First, for any $x \in \mathbb{H}_\Sigma \cap C$: $(v+d)(x) = \bot$ if $v(x) = \bot$; otherwise $(v+d)(x) = v(x) + d$. Second, for any $x \in \mathbb{P}_\Sigma \cap C$: $(v+d)(x) = v(x) - d$ if $v(x) \neq \bot$; otherwise $(v+d)(x) = \bot$. For a valuation $v \in \mathcal{V}(C)$, and a clock $x \in C$, we write $v[x := 0]$ the valuation that matches v on every clock $x' \neq x$ and such that $v(x) = 0$.

An *atomic clock constraint* over the set of clocks C is either true or a formula of the form $x \sim c$, where $x \in C$, $c \in \mathbb{N}$, and $\sim \in \{<, >, =\}$. A *clock constraint* is a Boolean combination of atomic clock constraints. We denote by $\mathsf{Constr}(C)$ the set of all clock constraints ranging over the set of clocks C. We say that a valuation v satisfies a clock constraint ψ, denoted $v \models \psi$ according to the following rules: $v \models \mathsf{true}$; $v \models x \sim c$ iff $v(x) \neq \bot$ and $v(x) \sim c$; $v \models \neg\psi$ iff $v \not\models \psi$; $v \models \psi_1 \vee \psi_2$ iff $v \models \psi_1$ or $v \models \psi_2$. We say that a timed word θ satisfies a clock constraint ψ at position $i \geq 0$, denoted $(\theta, i) \models \psi$ iff $\mathsf{Val}_\theta^i \models \psi$.

Alternating event clock automata Let X be finite set. A *positive Boolean formula* over X is Boolean formula generated by:

$$\varphi \quad ::= \quad a \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \mathsf{true} \mid \mathsf{false}$$

with $a \in X$, and φ_1, φ_2 positive Boolean formulas. We denote by $\mathcal{B}^+(X)$ the set of all positive Boolean formulas on X. A set $Y \subseteq X$ *satisfies* a positive Boolean formula $\varphi \in \mathcal{B}^+(X)$, denoted $Y \models \varphi$ if and only if replacing each $y \in Y$ by true and each $x \in X \setminus Y$ by false in φ, and applying the standard interpretation for $\vee$ and $\wedge$ yields a formula which is equivalent to true. For example, $\varphi = (q_1 \wedge q_2) \vee q_3$ is a positive Boolean formula on $\{q_1, q_2, q_3\}$. Clearly, $\{q_1, q_2\} \models \varphi$, $\{q_2, q_3\} \models \varphi$, but $\{q_1\} \not\models \varphi$. Given a set X, and a positive Boolean formula $\varphi \in \mathcal{B}^+(X)$, we denote by $\tilde{\varphi}$ the *dual of* φ, which is the positive Boolean formula obtained from φ by swapping the $\vee$ and $\wedge$ operators, as well as the true and false values.

An *alternating event-clock automaton* (AECA) is a tuple $A = \langle Q, q_{in}, \Sigma, \delta, \alpha \rangle$, where Q is a finite set of locations, $q_{in} \in Q$ is the initial location, Σ is a finite alphabet, $\delta : Q \times \Sigma \times \mathsf{Constr}(\mathbb{C}_\Sigma) \mapsto \mathcal{B}^+(Q)$ is a partial function, and α is the acceptance condition, which can be:

1. either a *Büchi acceptance condition*; in this case, $\alpha \subseteq Q$,

2. or a *co-Büchi acceptance condition*; in this case, $\alpha \subseteq Q$,

3. or a K*-co-Büchi acceptance condition*, for some $K \in \mathbb{N}$; in this case, $\alpha \subseteq Q$,

4. or a *parity condition*; in this case, $\alpha : Q \mapsto \mathsf{Colours}$, where $\mathsf{Colours} \subseteq \mathbb{N}$ is a *finite* set of *priorities*.

Moreover, δ respects the following conditions:

(A_1) For every $q \in Q$, $\sigma \in \Sigma$, $\delta(q, \sigma, \psi)$ is defined for only finitely many ψ.

(A_2) For every $q \in Q$, $\sigma \in \Sigma$, $v \in \mathcal{V}(\mathbb{C}_\Sigma)$ there exists one and only one $\psi \in \mathsf{Constr}(\mathbb{C}_\Sigma)$ s.t. $v \models \psi$ and $\delta(q, \sigma, \psi)$ is defined.

Runs and accepted languages Runs of AECA are formalised by *trees*. A *tree* T is a prefix closed set $T \subseteq \mathbb{N}^*$. The elements of T are called *nodes*, and the *root* of the tree is the empty sequence ε. For every $x \in T$, the nodes $x \cdot c \in T$, for $c \in \mathbb{N}$ are the *children* of x, and x is the (unique) *father* of all the nodes $x \cdot c$. A node with no child is a *leaf*. We refer to the length $|x|$ of x as its *level* in the tree. A *branch* in the tree T is a sequence of nodes $\pi \subseteq T$ such that $\varepsilon \in \pi$, and for every $x \in \pi$, either x is a leaf, or there is a unique $c \in \mathbb{N}$ such that $x \cdot c \in \pi$. An X*-labelled tree* is a pair $\langle T, \ell \rangle$ where $\ell : T \to X$ is a labelling function of the nodes, that associates a label from X to each node of T. We extend the function ℓ to (finite or infinite) branches: given a branch $\pi = n_1 n_2 \cdots n_j \cdots$ of T, we let $\ell(\pi)$ be the sequence $\ell(n_1)\ell(n_2)\cdots\ell(n_j)\cdots$ Let $A = \langle Q, q_{in}, \Sigma, \delta, \alpha \rangle$ be an AECA, and θ be an timed word on Σ. Then, a Q-labelled tree $R = \langle T, \ell \rangle$ is a *run of* A *on* θ iff the following hold:

- $\ell(\varepsilon) = q_{in}$,

- for all $x \in T$, there exists a set $S \subseteq Q$ s.t. (i) $q \in S$ iff x has a child $x \cdot c \in T$ with $\ell(x \cdot c) = q$ and (ii) $S \models \delta(\ell(x), w_{|x|}, \psi_x)$, where $\psi_x \in \mathsf{Constr}(\mathbb{C}_\Sigma)$ is the unique clock constraint s.t. $\delta(\ell(x), w_{|x|}, \psi_x)$ is defined and $(\theta, |x|) \models \psi_x$.

Let $R = \langle T, \ell \rangle$ be a run and $x \in T$. We note R_x the sub-run rooted at node x. A run $R = \langle T, \ell \rangle$ is *memoryless* if for all levels $i \in \mathbb{N}$, for all $x, y \in T$ such that $|x| = |y| = i$ and $\ell(x) = \ell(y)$, the sub-runs $R_x = \langle T_x, \ell_x \rangle$ and $R_y = \langle T_y, \ell_y \rangle$ are isomorphic.

Let $\langle T, \ell \rangle$ be an X-labelled tree, and let π be a branch of T. We let $\mathsf{Occ}_\pi : X \to \mathbb{N} \cup \{\infty\}$ be the function that associates, to any element of X, its number of occurrences in π. We further let $\mathsf{Inf}(\pi) = \{x \in X \mid \mathsf{Occ}_\pi(x) = \infty\}$. Let A be an AECA with set of locations Q and acceptance condition α, and $R = \langle T, \ell \rangle$ be a run of A. Then, R is an *accepting run* iff one of the following holds: α is a

- *Büchi condition*, and for all branches $\pi \subseteq T$, $\mathsf{Inf}(\pi) \cap \alpha \neq \varnothing$,
- *co-Büchi condition*, and for all branches $\pi \subseteq T$, $\mathsf{Inf}(\pi) \cap \alpha = \varnothing$,
- *K-co-Büchi condition*, and for all branches $\pi \subseteq T$, $\sum_{q \in \alpha} \mathsf{Occ}_\pi(q) \leq K$,
- *parity condition*, and for all branches $\pi \subseteq T$, $\max\{\alpha(q) \mid q \in \mathsf{Inf}(\pi)\}$ is even.

A timed word θ is *accepted* by an AECA A iff there exists an accepting run of A on θ. We denote by $\mathsf{L}(A)$ the *language* of A, i.e. $\mathsf{L}(A) = \{\theta \mid \theta \text{ is accepted by } A\}$, and by $\mathsf{L}(A)_{td}$ the time diverging language accepted by A, i.e. $\mathsf{L}(A)_{td} = \{\theta \mid \theta \in \mathsf{T}\Sigma^\omega_{\mathrm{td}} \; and \; \theta \text{ is accepted by } A\}$.

For readability, we often refer to the language of an automaton A with *co-Büchi acceptance condition* as $\mathsf{L}_{\mathsf{coB}}(A)$. Similarly, we use $\mathsf{L}_{\mathsf{B}}(A)$ to denote the accepted language of an automaton A with *Büchi acceptance condition*, $\mathsf{L}_{\mathsf{KcoB}}(A)$ in the case of an automaton A with *K-co-Büchi acceptance condition*, and $\mathsf{L}_{\mathsf{P}}(A)$ for an automaton A with *parity acceptance condition*.

Finally, let $A = \langle Q, q_{in}, \Sigma, \delta, \alpha \rangle$ be an AECA with Büchi acceptance condition. The *dual of* A, denoted $\tilde{A}$ is defined as the AECA $\left\langle Q, q_{in}, \Sigma, \tilde{\delta}, \alpha \right\rangle$ with co-Büchi acceptance condition, where for any $q \in Q$, $\sigma \in \Sigma$ and $\psi \in \mathsf{Constr}(\mathbb{C}_\Sigma)$, $\tilde{\delta}(q, \sigma, \psi)$ is equal to $\widetilde{\delta(q, \sigma, \psi)}$ iff $\delta(q, \sigma, \psi)$ is defined. It is easy to check that $\mathsf{L}_{\mathsf{coB}}(\tilde{A}) = \mathsf{T}\Sigma^\omega \setminus \mathsf{L}_{\mathsf{B}}(A)$.

Remark 36 (Time divergence) *It is easy to see that* $\mathsf{L}_{\mathsf{coB}}(\tilde{A})_{td} = \mathsf{T}\Sigma^\omega_{\mathrm{td}} \setminus \mathsf{L}_{\mathsf{B}}(A)$.

Syntactic restrictions Let us now define syntactic restrictions of AECA. Let $A = \langle Q, q_{in}, \Sigma, \delta, \alpha \rangle$ be an AECA. Then:

1. If, for any $q \in Q$, $\sigma \in \Sigma$ and $\psi \in \mathsf{Constr}(\mathbb{C}_\Sigma)$: $\delta(q, \sigma, \psi)$ is either undefined or a purely disjunctive formula, then A is a *non-deterministic event-clock automaton* (ECA for short).

2. If, for any $q \in Q$, $\sigma \in \Sigma$ and $\psi \in \mathsf{Constr}(\mathbb{C}_\Sigma)$: $\delta(q, \sigma, \psi)$ is either undefined or a purely conjunctive formula, then A is an *universal event-clock automaton* (UECA for short).

3. If, for any $q \in Q$, $\sigma \in \Sigma$ and $\psi \in \mathsf{Constr}(\mathbb{C}_\Sigma)$: $\delta(q, \sigma, \psi)$ is either undefined, or $\delta(q, \sigma, \psi) \in Q$, then A is a *deterministic event-clock automaton* (DECA for short).

4. If, for any $q \in Q$, $\sigma \in \Sigma$ and $\psi \in \mathsf{Constr}(\mathbb{C}_\Sigma)$: either $\delta(q, \sigma, \psi)$ is undefined or $\psi \in \mathsf{Constr}(\mathbb{H}_\Sigma)$, then A is a *past event-clock automaton* (PastECA for short).

5. If, for any $q \in Q$, $\sigma \in \Sigma$: $\delta(q, \sigma, \mathsf{true})$ is defined, then A is an *alternating word automaton* (AWA for short). In this case, since the third parameter of δ is always true, we omit it. We refer to such automata as *untimed* word automata. We use the shorthands NWA and DWA to refer to non-deterministic and deterministic (untimed) word automata.

Given a NECA A and a timed word θ on Σ, if there exists an accepting run $R = \langle T, \ell \rangle$ of A on θ, then it is easy to see that there exists an accepting run with one branch π.

We denote such a run by the sequence $q_{in}, (\sigma_0, \tau_0), q_1, (\sigma_1, \tau_1), \cdots, q_j, (\sigma_j, \tau_j), \cdots$ where $q_{in} q_1 \cdots q_j \cdots$ is the label $\ell(\pi)$ of the single branch π of T.

6.5.2 Regionalization of alternating event clock automata

We define the notion of equivalence for the valuations of clocks defined above. We will use it to regionalize the timed words and the alternating event clock automata.

Equivalences for event-clock valuations We define two notions of equivalence for valuations of clocks, the former called *weak equivalence* and the latter called *strong equivalence*. The notion of weak equivalence applies to valuations for both history clocks and prophecy clocks, while the notion of strong equivalence applies to valuations for history clocks only. They are defined as follows.

Let $C \subseteq \mathbb{H}_\Sigma \cup \mathbb{P}_\Sigma$, and let $cmax \in \mathbb{N}$. Two valuations $v_1, v_2 \in \mathcal{V}(C)$ are *weakly equivalent*, noted $v_1 \sim_{cmax} v_2$, iff the following two conditions are satisfied:

(C_1) $\forall x \in C$: $v_1(x) = \bot$ iff $v_2(x) = \bot$;

(C_2) $\forall x \in C$: either $v_1(x) > cmax$ and $v_1(x) > cmax$, or $\lceil v_1(x) \rceil = \lceil v_2(x) \rceil$ and $\lfloor v_1(x) \rfloor = \lfloor v_2(x) \rfloor$.

We note $[v]_{\sim cmax}$ the weak equivalence class of v. We note $\mathsf{wReg}\,(C, cmax)$ the finite set of equivalence classes of the relation $\sim_{cmax}$, and call them *weak regions*.

Lemma 37 *Let $C \subseteq \mathbb{H}_\Sigma \cup \mathbb{P}_\Sigma$, and let $cmax \in \mathbb{N}$. Two valuations $v_1, v_2 \in \mathcal{V}(C)$ are weakly equivalent iff for all $\psi \in \mathsf{Constr}\,(C, cmax)$: $v_1 \models \psi$ iff $v_2 \models \psi$.*

Let $C \subseteq \mathbb{H}_\Sigma$, and let $cmax \in \mathbb{N}$. Two valuations $v_1, v_2 \in \mathcal{V}(C)$ are *strongly equivalent*, noted $v_1 \approx_{cmax} v_2$, iff conditions C_1 and C_2 are satisfied and additionally:

(C_3) $\forall x_1, x_2 \in C$: $\lceil v_1(x_1) \rceil - v_1(x_1) \leq \lceil v_1(x_2) \rceil - v_1(x_2)$ iff $\lceil v_2(x_1) \rceil - v_2(x_1) \leq \lceil v_2(x_2) \rceil - v_2(x_2)$.

We note $[v]_{\approx cmax}$ the strong equivalence class of v, we note $\mathsf{Reg}\,(C, cmax)$ the finite set of equivalence classes of the relation $\approx_{cmax}$, and we call them *strong regions*, or simply *regions*. Note that our notion of strong equivalence for valuations of history clocks is an adaptation of the classical notion of clock equivalence defined for timed automata [Alur and Dill, 1994], hence it is a time-abstract bisimulation. For any region $r \in \mathsf{Reg}\,(C, cmax)$, we say that $r' \in \mathsf{Reg}\,(C, cmax)$ is a *time-successor* of r (written $r \leq_{\mathsf{t.s.}} r'$) if and only if for any valuation $v \in r$, there is some $t \in \mathbb{R}^{\geq 0}$ such that $v + t \in r'$. Note that the relation $\leq_{\mathsf{t.s.}}$ is a partial order over $\mathsf{Reg}\,(C, cmax)$. A region $r \in \mathsf{Reg}\,(C, cmax)$ is *initial* if, for all $v \in r$, for all (history) clock $x \in C$: $v(x) = \bot$. Note that the initial region is unique and denoted r^C_{in} (when C is clear from the context we denote it by r_{in}). Finally, for all $r \in \mathsf{Reg}\,(C, cmax)$ and all $x \in C$, we note $r[x := 0]$ the region s.t. for all $v \in r[x := 0]$, there is $v' \in r$ with $v'[x := 0] = v$.

Region automaton Given a set of history clocks $C \subseteq \mathbb{H}_\Sigma$ and $cmax \in \mathbb{N}$, the *region automaton* $\mathsf{RegAut}\,(C, cmax) = \langle \mathsf{Reg}\,(C, cmax) \cup \{\bot\}, r^C_{\text{in}}, \Sigma^R, \delta^R, \alpha \rangle$, is a DWA where $\Sigma^R = \Sigma \times \mathsf{Reg}\,(C, cmax)$ and $\alpha = \mathsf{Reg}\,(C, cmax)$ is a Büchi acceptance condition. The transition relation δ^R is such that for all $r, r' \in \mathsf{Reg}\,(C, cmax)$, and for all $\sigma \in \Sigma$:

- $\delta^R(r, (\sigma, r')) = r'[\overleftarrow{x_\sigma} := 0]$ if $r \leq_{\mathsf{t.s.}} r'$, otherwise $\delta^R(r, (\sigma, r')) = \bot$,

- $\delta^R(\bot, (\sigma, r')) = \bot$.

Regionalizations of a timed word Given $C \subseteq \mathbb{C}_\Sigma$, $cmax \in \mathbb{N}$, and a timed word $\theta = (\sigma_0, \tau_0)(\sigma_1, \tau_1)\cdots \in \mathsf{T}\Sigma^\omega \cup \mathsf{T}\Sigma^*$, let v_i be the restriction of $\mathsf{Val}_{\mathsf{i}}^\theta$ to the set of clocks C. We define the *weak region word associated to* θ, denoted $\mathsf{wrg}(C, cmax, \theta)$ as the (untimed) word $(\sigma_0, [v_0]_{\sim cmax})(\sigma_1, [v_1]_{\sim cmax})\cdots$ over $\Sigma \times \mathsf{wReg}\,(C, cmax)$. Intuitively, $\mathsf{wrg}(C, cmax, \theta)$ describes, along with the sequence of letters, the sequence of weak regions visited by θ. If $C \subseteq \mathbb{H}_\Sigma$, we also define the *(strong) region word associated to* θ, denoted $\mathsf{rg}(C, cmax, \theta)$ as the (untimed) word $(\sigma_0, [v_0]_{\approx cmax})(\sigma_1, [v_1]_{\approx cmax})\cdots$ over $\Sigma \times \mathsf{Reg}\,(C, cmax)$. We extend wrg and rg to set of words L: $\mathsf{wrg}(C, cmax, L) = \{\mathsf{wrg}(C, cmax, \theta) \mid \theta \in L\}$ and $\mathsf{rg}(C, cmax, L) = \{\mathsf{rg}(C, cmax, \theta) \mid \theta \in L\}$.

Proposition 38 *For all set of clocks* $C \subseteq \mathbb{H}_\Sigma$ *and* $cmax \in \mathbb{N}$*:* $\mathsf{L_B}(\mathsf{RegAut}\,(C, cmax)) = \mathsf{rg}(C, cmax, \mathsf{T}\Sigma^\omega)$.

Remark 39 (Time divergence) *We can extend the definition of* region automaton *to obtain an automaton* $\mathsf{RegAut_{td}}\,(C, cmax)$ *that accepts all the infinite words over* $\Sigma \times \mathsf{Reg}\,(C, cmax)$ *associated to* diverging *timed words. To achieve this, we must use a* generalized Büchi acceptance condition *that guarantees time divergence on the regions (see [Alur and Dill, 1994] for the details). Then,* $\mathsf{L}(\mathsf{RegAut_{td}}\,(C, cmax)) = \mathsf{rg}(C, cmax, \mathsf{T}\Sigma^\omega_{\mathrm{td}})$.

Regionalizations of an AECA Let $A = \langle Q, q_{in}, \Sigma, \delta, \alpha\rangle$ be an AECA, let $C \subseteq \mathbb{C}_\Sigma$ be the set of clocks and $cmax \in \mathbb{N}$ be the maximal constant appearing in A. We define the *weak regionalization of* A as the AWA $\mathsf{wRg}(A) = \langle Q, q_{in}, \Sigma \times \mathsf{wReg}\,(C, cmax)\,, \delta', \alpha\rangle$ s.t. for all $q \in Q$, and $(\sigma, r) \in \Sigma \times \mathsf{wReg}\,(C, cmax)$: $\delta'(q, (\sigma, r)) = \delta(q, \sigma, \psi)$ where ψ is the unique constraint such that $\delta(q, \sigma, \psi)$ is defined and $v \models \psi$ for all $v \in r$.

Let $A = \langle Q, q_{in}, \Sigma, \delta, \alpha\rangle$ be a PastECA, let $C \subseteq \mathbb{H}_\Sigma$ be the set of history clocks and $cmax \in \mathbb{N}$ be the maximal constant appearing in A. We define the *(strong) regionalization of* A as the AWA $\mathsf{Rg}(A) = \langle Q, q_{in}, \Sigma \times \mathsf{Reg}\,(C, cmax)\,, \delta', \alpha\rangle$ s.t. for all $q \in Q$, and $(\sigma, r) \in \Sigma \times \mathsf{Reg}\,(C, cmax)$: $\delta'(q, (\sigma, r)) = \delta(q, \sigma, \psi)$ where ψ is the unique constraint such that $\delta(q, \sigma, \psi)$ is defined and $v \models \psi$ for all $v \in r$.

The following lemma links runs in an AECA, and its weak and (strong) regionalization (when A is a PastECA).

Lemma 40 *Let A be an* AECA. *For every timed word $\theta \in \mathsf{T}\Sigma^\omega$, $R = \langle T, \ell \rangle$ is an accepting run tree of A over θ iff it is an accepting run tree of* $\mathsf{wRg}(A)$ *over* $\mathsf{wrg}(C, cmax, \theta)$. *Moreover, if A is a* PastECA, *$R = \langle T, \ell \rangle$ is an accepting run tree of A over θ iff it is an accepting run tree of* $\mathsf{Rg}(A)$ *over* $\mathsf{rg}(C, cmax, \theta)$.

The following lemma states that, for all PastECA A, the words accepted by both $\mathsf{Rg}(A)$ and by RegAut $(C, cmax)$ are exactly the (strong) regionalizations of the timed words accepted by A (whatever the acceptance condition of A is):

Lemma 41 *For all* PastECA $A = \langle Q, q_{in}, \Sigma, \delta, \alpha \rangle$, *with set of clocks $C \subseteq \mathbb{H}_\Sigma$ and maximal constant $cmax$:* $\mathsf{L}(\mathsf{Rg}(A)) \cap \mathsf{L}_\mathsf{B}(\mathsf{RegAut}\,(C, cmax)) = \mathsf{rg}(C, cmax, \mathsf{L}(A))$.

Proof Let θ be a word in $\mathsf{L}(A)$. Then there is an accepting run $R = \langle T, \ell \rangle$ of A over θ. By Proposition 38, $\mathsf{rg}(C, cmax, \theta) \in \mathsf{L}_\mathsf{B}(\mathsf{RegAut}\,(C, cmax))$. By Lemma 40, R is also a accepting run of $\mathsf{Rg}(A)$ over $\mathsf{rg}(C, cmax, \theta)$. Thus, $\mathsf{rg}(C, cmax, \theta) \in \mathsf{L}(\mathsf{Rg}(A))$.

Conversely, let w be a word in $\mathsf{L}(\mathsf{Rg}(A)) \cap \mathsf{L}_\mathsf{B}(\mathsf{RegAut}\,(C, cmax))$. Since $w \in \mathsf{L}_\mathsf{B}(\mathsf{RegAut}\,(C, cmax))$, by Proposition 38, there is $\theta \in \mathsf{T}\Sigma^\omega$ such that $\mathsf{rg}(C, cmax, \theta) = w$. Let $R = \langle T, \ell \rangle$ be an accepting run of $\mathsf{Rg}(A)$ over w. By Lemma 40, R is also a accepting run of A over θ and thus $\theta \in \mathsf{L}(A)$. □

Remark 42 (Time divergence) *If we restrict our attention to diverging timed words, then:* $\mathsf{L}(\mathsf{Rg}(A)) \cap \mathsf{L}(\mathsf{RegAut}_{\mathsf{td}}\,(C, cmax)) = \mathsf{rg}(C, cmax, \mathsf{L}(A)_{td})$

6.5.3 Rank contruction for alternating event clock automata

In this subsection, we show how to complement AECA with Büchi acceptance condition. This procedure allows us to solve the universality and language inclusion problems for NECA with Büchi acceptance condition without resorting to determinization procedures (like the one defined by Safra [1988]) that are resistant to efficient implementation.

We start by showing how to transform a co-Büchi acceptance condition into a Büchi condition when considering AECA. For that, we need the existence of memoryless runs:

Lemma 43 *Let A be an* AECA *with co-Büchi acceptance condition. For all timed words θ such that $\theta \in \mathsf{L}_{\mathsf{coB}}(A)$: A has an accepting memoryless run on θ.*

Proof Let C be the set of clocks and $cmax$ be the maximal constant of A. Let θ be a timed word accepted by A. By Lemma 40, $\mathsf{wrg}(C, cmax, \theta)$ is accepted by the AWA $\mathsf{wRg}(C, cmax, A)$. Let $R = \langle T, \ell \rangle$ be an accepting run of $\mathsf{wRg}(C, cmax, A)$ on $\mathsf{wrg}(C, cmax, \theta)$. By the result of Emerson and Jutla [1991][Theorem 4.4], we can make the hypothesis that R is memoryless. By Lemma 40, R is an accepting run of A on θ. □

The memoryless property of accepting runs in AECA with co-Büchi acceptance condition allows us to represent those runs as DAGs where isomorphic subtrees are merged. Formally, we associate to every memoryless run $R = \langle T, \ell \rangle$ of $A = \langle Q, q_{in}, \Sigma, \delta, \alpha \rangle$ the DAG $G_R = \langle V, E \rangle$, where the set of vertices $V \subseteq Q \times \mathbb{N}$ represents the labels of the nodes of R at each level. Formally, $(q, l) \in V$ if and only if there is a node $x \in T$ such that $|x| = l$ and $\ell(x) = q$. The set of edges $E \subseteq \bigcup_{l \geq 0}(Q \times \{l\}) \times (Q \times \{l+1\})$ relates the nodes of one level to their children. Formally, $((q, l), (q', l+1)) \in E$ if and only if there exists some node $x \in T$ and $c \in \mathbb{N}$ such that $x \cdot c \in T$ and $|x| = l$, $\ell(x) = q$, and $\ell(x \cdot c) = q'$. Note that the width of the DAG is bounded by $|Q|$.

Now, we can apply results of Kupferman and Vardi [2001] that characterize the structure of accepting runs of *alternating automata* with co-Büchi acceptance condition. For that we need some additional notations. For $k \in \mathbb{N}$ we write $[k]$ for the set $\{0, 1, \ldots, k\}$ and $[k]^{\text{odd}}$ for the set of odd elements of $[k]$. The following lemma is adapted from [Kupferman and Vardi, 2001]:

Lemma 44 *Let A be an* AECA *with n locations and co-Büchi accepting condition α. The vertices of the DAG G_R associated to a memoryless accepting run R of A can be labelled by a ranking function $f : V \to [2n]$ having the following properties:*

(P_1) *for $(q, l) \in Q \times \mathbb{N}$, if $f(q, l)$ is odd, then $q \notin \alpha$,*

(P_2) *for (q, l) and (q', l') such that (q', l') is reachable from (q, l), $f(q', l') \leq f(q, l)$,*

(P_3) *in every infinite path π in G_R, there exists a node (q, l) such that $f(q, l)$ is odd and, for all (q', l') in π reachable from (q, l): $f(q', l') = f(q, l)$.*

We use this ranking function to justify the transformation of an AECA with co-Büchi acceptance condition into an AECA with Büchi acceptance condition.

Let $A = \langle Q, q_{in}, \Sigma, \delta, \alpha \rangle$ be an AECA with co-Büchi acceptance condition, and let $|Q| = n$. We define the AECA $\mathsf{Rank}(A)$ as $\langle Q', q'_{in}, \Sigma, \delta', \alpha' \rangle$ with Büchi acceptance condition, where $Q' = Q \times [2n]$, $q'_{in} = (q_{in}, 2n)$, and δ' is defined using the auxiliary function (we use the notations of [Kupferman and Vardi, 2001]): $\mathsf{release} : \mathcal{B}^+(Q) \times [2n] \to \mathcal{B}^+(Q')$, which maps a formula $\phi \in \mathcal{B}^+(Q)$ and an integer $i \in [2n]$ to a formula obtained from ϕ by replacing each atom $q \in Q$ by the disjunction $\bigvee_{j \leq i}(q, j)$. Then, for any $(q, i) \in Q'$, $\sigma \in \Sigma$ and $\psi \in \mathsf{Constr}(\mathbb{C}_\Sigma)$ such that $\delta(q, \sigma, \psi)$ is defined,

$$\delta'((q,i),\sigma,\psi) = \begin{cases} \mathsf{release}(\delta(q,\sigma,\psi),i) & \text{if } q \notin \alpha \text{ or } i \text{ is even,} \\ \mathsf{false} & \text{if } q \in \alpha \text{ and } i \text{ is odd.} \end{cases}$$

Finally, $\alpha' = Q \times [2n]^{\text{odd}}$ is a Büchi acceptance condition.

Remark that, by condition A_2 of the definition of the transition relation in AECA, for all $q \in Q$, $\sigma \in \Sigma$ and valuation $v \in \mathcal{V}(C)$, there is exactly one clock constraint ψ such that $\delta(q, \sigma, \psi)$ is defined and $v \models \psi$. Thus, by construction of $\mathsf{Rank}(A)$, for all $q \in Q$, $i \in [n]$ and valuation $v \in \mathcal{V}(C)$, there is exactly one clock constraint ψ such that $\delta'((q, i), w_k, \psi)$ is defined and $v \models \psi$. Thus, δ' is well-formed. Let us establish the relationship between the accepted languages of A and $\mathsf{Rank}(A)$.

Proposition 45 *For all* AECA A *with co-Büchi condition:* $\mathsf{L_B}(\mathsf{Rank}(A)) = \mathsf{L_{coB}}(A)$.

Proof Let $\theta = (\sigma, \tau) \in \mathsf{T}\Sigma^\omega$ be a timed word in $\mathsf{L_B}(\mathsf{Rank}(A))$ and let us show that $\theta \in \mathsf{L_{coB}}(A)$. Let $R' = \langle T, \ell' \rangle$ be an accepting run of $\mathsf{Rank}(A)$ on θ. Consider $R = \langle T, \ell \rangle$ where for all $x \in T$, $\ell(x) = q$ if $\ell'(x) = (q, j)$ for some rank j. By definition of $\mathsf{Rank}(A)$, R is a run of A on θ. Let us now show that it is an accepting run of A. As R' is accepting for $\mathsf{Rank}(A)$, we know that every branch has the following property: from some level $i \in \mathbb{N}$, the rank j is not changing anymore. This is because the definition of the transition function of $\mathsf{Rank}(A)$ requires the ranks to decrease along a path, while staying positive. Moreover, the acceptance condition imposes that this rank is odd. Let π be such a branch. As accepting locations of A are associated to odd ranks and cannot appear in runs of $\mathsf{Rank}(A)$ (it is forbidden by the transition relation), we know that the branch π in R visits only finitely many accepting locations and so it respects the acceptance condition of A.

Conversely, let $\theta \in \mathsf{L_{coB}}(A)$ and let us show that $\theta \in \mathsf{L_B}(\mathsf{Rank}(A))$. Let $R = (T, \ell)$

be an accepting run of A on θ. Now consider the tree $R' = (T, \ell')$, where ℓ' is s.t. $\ell(\varepsilon) = (q_{in}, 2n)$ and for all $x \in T$, $\ell'(x) = (\ell(x), f(x))$. Following properties P_1 and P_2 of Lemma 44, $R' = (T, \ell')$ is a run of $\mathsf{Rank}(A)$ over the timed word θ. Let π be a branch of R'. Then, property P_3 in Lemma 44 ensures that at some point, all the states in π are labelled by the same odd rank. Thus, any branch of R' visits infinitely often a state in $Q \times [2n]^{\text{odd}}$, and $\mathsf{Rank}(A)$ is accepting. □

Next, we show that the construction due to Miyano and Hayashi [1984] to transform an alternating Büchi automaton into a nondeterministic one can be easily adapted to AECA with Büchi acceptance condition.

Formally, given an AECA with Büchi acceptance condition $A = \langle Q, q_{in}, \Sigma, \delta, \alpha \rangle$, we define a NECA $\mathsf{MH}(A)$ as follows. For any $\sigma \in \Sigma$, for any $q \in Q$, let $\Phi_q^\sigma = \{\psi \in \mathsf{Constr}(\mathbb{C}_\Sigma) \mid \delta(q, \sigma, \psi) \text{ is defined}\}$. By condition A_1 of the definition of an AECA, Φ_q^σ is finite. We also define, for any $\sigma \in \Sigma$, for any subset $S \subseteq Q$, the set of formulas $\Psi_S^\sigma = \{\bigwedge_{q \in S} \psi_q \mid \psi_q \in \Phi_q^\sigma\}$. Intuitively, Ψ_S^σ contains all the conjunctions that contain exactly one conjunct from each set Φ_q^σ (for $q \in S$). Finally, for $S \subseteq Q$, $O \subseteq Q$, $\sigma \in \Sigma$, $\psi = \bigwedge_{q \in S} \psi_q \in \Psi_S^\sigma$, we let $P(S, O) = \{(S', O') \mid S' \models \bigwedge_{q \in S} \delta(q, \sigma, \psi_q), O' \subseteq S', O' \models \bigwedge_{q \in O} \delta(q, \sigma, \psi_q)\}$ if $O \neq \varnothing$, and $P(S, \varnothing) = \{(S', S') \mid S' \models \bigwedge_{q \in S} \delta(q, \sigma, \psi_q)\}$.

Then, we define $\mathsf{MH}(A)$ as the AECA $\langle 2^Q \times 2^Q, (\{q_{in}\}, \varnothing), \Sigma, \delta', 2^Q \times \{\varnothing\} \rangle$ with Büchi acceptance condition where, for any $(S, O) \in 2^Q \times 2^Q$, for any $\sigma \in \Sigma$, for any $\psi \in \Psi_S^\sigma$: $\delta'((S, O), \sigma, \psi) = \bigvee_{(S', O') \in P(S,O)} (S', O' \setminus \alpha)$ (and δ' is undefined otherwise). Remark that, by conditions A_1 and A_2, Ψ_S^σ is a finite set, and for any valuation v, there is exactly one $\psi \in \Psi_S^\sigma$ s.t. $v \models \psi$. Hence, δ' respects the definition of the transition relation of an AECA. The next proposition proves the correctness of the construction.

Proposition 46 *For all* AECA A *with Büchi condition:* $\mathsf{L_B}(\mathsf{MH}(A)) = \mathsf{L_B}(A)$.

Proof Assume $A = \langle Q, q_{in}, \Sigma, \delta, \alpha \rangle$. Let θ be a timed word in $\mathsf{L_B}(A)$ and $R = \langle T, \ell \rangle$ be an accepting run of A over θ. Then, let $\rho = (\{q_{in}\}, \varnothing), (\sigma_0, \tau_0), (S_1, O_1), (\sigma_1, \tau_1), \cdots$ be the sequence such that, for all $i \in \mathbb{N}$: (i) $S_i = \{q \mid \exists x \in T, |x| = i, \ell(x) = q\}$ and (ii) $O_i = S_i \setminus \alpha$ if $O_{i-1} = \varnothing$; $O_i = \{q \mid \exists x \cdot c \in T, |x \cdot c| = i, \ell(x \cdot c) = q, \ell(x) \in O_{i-1}\} \cap (Q \setminus \alpha)$ otherwise (with the convention that $O_0 = \varnothing$). It is easy to see that, as in the original construction of Miyano and Hayashi [1984], ρ is an accepting run of $\mathsf{MH}(A)$ over θ.

Conversely, given a run $(\{q_{in}\}, \varnothing)(\sigma_0, \tau_0)(S_1, O_1)(\sigma_1, \tau_1)(S_2, O_2)\cdots$ of $\mathsf{MH}(A)$, we consider a labelled tree $\langle T, \ell\rangle$ s.t. (i) $\ell(\varepsilon) = q_{in}$ and (ii) for any $x \in T$: $\{\ell(x \cdot i) \mid i \in \mathbb{N}\} \subseteq S_{|x|+1}$ and $\{\ell(x \cdot i) \mid i \in \mathbb{N}\} \models \delta(\ell(x), \sigma_{|x|}, \psi)$, where ψ is the unique constraint s.t. $\delta(\ell(x), \sigma_{|x|}, \psi)$ is defined and $(\theta, |x|) \models \psi$. Clearly, R is an accepting run tree of A over θ. □

6.5.4 Applications

Let us show how to apply these constructions to complement an NECA.

Given a NECA $A = \langle Q, q_{in}, \Sigma, \delta, \alpha\rangle$ with Büchi acceptance condition, we first construct its dual $\widetilde{A}$ which is thus a UECA with co-Büchi acceptance condition s.t. $\mathsf{L}_{\mathsf{coB}}(\widetilde{A}) = \mathsf{T}\Sigma^\omega \setminus \mathsf{L}_{\mathsf{B}}(A)$. Then, thanks to Proposition 45 and Proposition 46, it easy to check that $\mathsf{MH}(\mathsf{Rank}(\widetilde{A}))$ is a NECA with Büchi condition s.t. $\mathsf{L}_{\mathsf{B}}(\mathsf{MH}(\mathsf{Rank}(\widetilde{A}))) = \mathsf{T}\Sigma^\omega \setminus \mathsf{L}_{\mathsf{B}}(A)$.

This construction can be applied to solve the *language inclusion* and *language universality* problems, because $\mathsf{L}_{\mathsf{B}}(A)$ is universal iff $\mathsf{T}\Sigma^\omega \setminus \mathsf{L}_{\mathsf{B}}(A)$ is empty and $\mathsf{L}_{\mathsf{B}}(B) \subseteq \mathsf{L}_{\mathsf{B}}(A)$ iff $\mathsf{L}_{\mathsf{B}}(B) \cap (\mathsf{T}\Sigma^\omega \setminus \mathsf{L}_{\mathsf{B}}(A))$ is empty.

Remark 47 (Time divergence) *All the constructions presented above are valid if we consider the time divergent semantics. Indeed,* $\mathsf{L}(A)_{td} \subseteq \mathsf{L}(B)_{td}$ *if and only if* $(\mathsf{L}(A) \cap \mathsf{L}(B)) \cap \mathsf{T}\Sigma^\omega_{\mathrm{td}} = \varnothing$

Remark 48 (Efficient implementation) *In [Doyen and Raskin, 2010], it is shown how to use subsumption to implement efficient emptiness test for automata defined by the Miyano and Hayashi construction without explicitly constructing them. Those methods can be readily extended to the case of event-clock automata.*

Chapter 7

Safraless Realizability Problem for Real Time Logics

This Chapter focuses on the realizability problem for real time logics. We introduce the definition of linear temporal logic synthesis, starting form the classical solutions that use Safra determinization and then defining Safraless approaches for LTL synthesis. In particular, we solve the realizability problem for a fragment of the Event Clocks Logic called $\mathsf{LTL}_{\triangleleft}$, providing an algorithm to solve the realizability problem for timed specifications through a reduction to a timed safety game problem. Finally, we show that this timed safety game problem can be solved using the model checking tool UPPAAL TIGA, illustrating this on a simple example.

7.1 A Brief of Game Theory

Game theory is a branch of applied mathematics that is used in the social sciences, most notably in economics, as well as in biology, engineering, political science, international relations, computer science, social psychology, philosophy and management. Game theory attempts to mathematically capture behavior in strategic situations, or games, in which an individual's success in making choices depends on the choices of others. The games studied in game theory are well-defined mathematical objects. A game consists of a set of players, a set of moves (or strategies) available to those players, and a specification of payoffs for each combination of strategies.

The mathematical theory of games was invented by von Neumann and Morgenstern [1944]. All situations in which at least one agent can only act to maximize his utility through anticipating (either consciously, or just implicitly in his behavior) the responses to his actions by one or more other agents is called a *game*. Agents involved in games are referred to as *players*.

Game theory has come to play an increasingly important role in logic and in computer science. Several logical theories have a basis in game semantics. In addition, computer scientists have used games to model interactive computations. Game semantics is an approach to formal semantics that grounds the concepts of truth or validity on game-theoretic concepts, such as the existence of a winning strategy for a player.

The simplest application of game semantics is to propositional logic. Each formula of this language is interpreted as a game between two players, known as the "Verifier" and the "Falsifier". The Verifier is given "ownership" of all the disjunctions in the formula, and the Falsifier is likewise given ownership of all the conjunctions. Each move of the game consists of allowing the owner of the dominant connective to pick one of its branches; play will then continue in that subformula, with whichever player controls its dominant connective making the next move. Play ends when a primitive proposition has been so chosen by the two players; at this point the Verifier is deemed the winner if the resulting proposition is true, and the Falsifier is deemed the winner if it is false. The original formula will be considered true precisely when the Verifier has a winning strategy, while it will be false whenever the Falsifier has the winning strategy.

Our main interest is the synthesis of controllers for reactive systems. A concise way to specify requirements on infinite executions is to use linear temporal logic (LTL). Its advantages include a compact, variable-free syntax and intuitive semantics which makes LTL suitable to be used in applications.

7.2 The Realizability Problem for LTL

Given a specification, the *Realizability Problem* (or *Program Synthesis*) is the automatic construction of a design that is guaranteed to be correct. If a system has been specified precisely, then it should be possible to generate the program automatically, avoiding the costs of separately developing a possibly incorrect system [Kupferman

et al., 2001].

In computer system design, there is a distinction between closed and open systems [Harel and Pnueli, 1985].

A closed system is a system where both program and user work together to find the required output and whose behavior is completely determined by the state of the system. An open system (or reactive system) is a system that interacts with its environment and whose behavior depends on this interaction; it assumes a hostile environment. As an example to closed and open systems, we can think of two drink-dispensing machines. One machine, which is a closed system, repeatedly boils water, makes an internal nondeterministic choice, and serves either coffee or tea. The second machine, which is an open system, repeatedly boils water, asks the environment to choose between coffee and tea, and deterministically serves a drink according to the external choice [Hoare, 1985].

Both machines induce the same infinite tree of possible executions. Nevertheless, while the behavior of the first machine is determined by internal choices solely, the behavior of the second machine is determined also by external choices, made by its environment. Formally, in a closed system, the environment cannot modify any of the system variables. In contrast, in an open system, the environment can modify some of the system variables.

Designing correct open systems is not an easy task. The design has to be correct with respect to any environment, and often there is much uncertainty regarding the environment [Fischer and Zuck, 1988].

Therefore, in the context of open systems, formal specification and verification of the design has great importance. Traditional formalisms for specification of systems relate the initial state and the final state of a system [Floyd, 1967; Hoare, 1969].

In 1977, Pnueli suggested temporal logics as a suitable formalism for reasoning about the correctness of nonterminating systems [Pnueli, 1977]. The breakthrough that temporal logics brought to the area of specification and verification arises from their ability to describe an ongoing interaction of a reactive module with its environment [Harel and Pnueli, 1985]. This ability makes temporal logics particularly appropriate for the specification of open systems.

Two possible views regarding the nature of time induce two types of temporal logics [Lamport, 1980].

In linear temporal logics, time is treated as if each moment in time has a unique possible future. Thus, linear temporal logic formulas are interpreted over linear sequences and we regard them as describing the interaction of the system with its environment along a single computation. In branching temporal logics, each moment in time may split into various possible futures. Accordingly, the structures over which branching temporal logic formulas are interpreted are infinite trees, and they describe the possible interactions of a system with its environment. In both paradigms, we can describe the design in some formal model, specify its required behavior with a temporal logic formula, and check formally that the model satisfies the formula.

The introduction of temporal logic gave rise to further developments in the area of synthesis of reactive systems [Jobstmann, 2007].

Emerson and Clarke [1982] and Manna and Wolper [1984] considered the problem for temporal specifications given in branching time logic CTL and linear temporal logic LTL, respectively. Both concluded that if a specification expressed by a formula φ is satisfiable, it is possible to construct a system that adheres to the specification using the model that satisfies φ. Due to the reduction to satisfiability the approaches are limited to constructing closed systems, which lead to systems that are only guaranteed to work correctly in cooperative environments, that is environments that help to satisfy φ.

On the other hand, to provide a solution for constructing open systems the best method used is Synthesis that aims to transform a specification into a system that is guaranteed to satisfy the specification. The theory behind synthesis of reactive systems is well established and goes back to Church [1962], who stated the Synthesis Problem using different fragments of restricted recursive arithmetic (S1S) as specification. The use of S1S for specifying the behavior of reactive systems is not convenient. For that reason there was an urgent need for new specification languages. A very successful proposal was the introduction of temporal logic [Emerson and Clarke, 1982; Pnueli, 1977], which is now widely used in the formal verification community. Specifying is easier in Linear Temporal Logic (LTL) [Manna and Pnueli, 1992a; Pnueli, 1977], which is also more suitable for compositional reasoning [Kupferman et al., 2000; Vardi, 2001].

LTL synthesis is the process of generating a reactive finite-state system from a formal specification written in linear temporal logic (LTL). The idea of synthesis is to automatically construct a functional correct system from a behavioral description of

the system.

In the late 80s, Pnueli and Rosner [1989] reconsidered the topic for LTL and provided a solution for constructing open systems. The key observation (also observed by Rabin [1972] is that even though the specification can be represented as infinite sequences (words) over the input and output signals, the solution to the synthesis problem is an infinite tree. Furthermore, Rosner [1992] proved that synthesis of LTL properties is 2EXPTIME-complete.

The first exponent derives from the translation of the LTL formula into a nondeterministic Büchi automaton. The second exponent is due to the determinization of the automaton.

Even though the idea of LTL synthesis is nearly fifty years old and the underlying theory is well established, it has not be adapted to practice yet. The first reason is that synthesis of LTL properties is 2EXPTIME-complete [Rosner, 1992].

The second is that the solution to LTL synthesis [Pnueli and Rosner, 1989] uses an intricate determinization construction [Safra, 1988] that is hard to implement and very hard to optimize. Thirdly, the solution to synthesis is not compositional and therefore does not reflect the usually iterative process of writing a complete specification. The bound introduced in the first reason is a lower bound as shown by Rosner [1992], so there are specifications for which the smallest correct system is doubly exponentially larger than the specification. Thus, the worst case complexity of verifying the specification is also 2EXPTIME in terms of the (full) specification. In combination with the second reason, however, the argument gains strength. For many specifications, a doubly-exponential blow up is not necessary, but can only be avoided through careful use of optimization techniques, which is hard to achieve in combination with Safra's algorithm.

Recently developed algorithms by Kupferman and Vardi [2005b] and by Kupferman et al. [2006], and Piterman et al. [2006] follow along completely different paths and give new hope for the synthesis problem.

The problem of automatic synthesis is usually formalized as a two-players game, for example in the work of Doyen et al. [2009] it is presented the definition of a two-players game, in which *Player 0* controls the execution of the system, and *Player 1* controls the execution of environment. The specification is encoded as the winning condition for Player 0 in the game. Roughly speaking, the behaviors of Player 1 repre-

sent all possible models for the system, and computing a winning strategy for Player 0 amounts to selecting one model which is guaranteed to be correct whatever the environment does.

7.2.1 Realizability as infinite game

A game is composed of an arena and a winning condition [Mazala, 2001].

An arena is a triple $A = (V_0, V_1, E)$, where V_0 is a set of 0-vertices, V_1 a set of 1-vertices, disjoint from V_0, and $E \subseteq (V_0 \cup V1) \times (V_0 \cup V1)$ is the edge relation, sometimes also called the set of moves. The union of V_0 and V_1 is denoted V.

Observe that with this notation the requirement for the edge relation reads$E \subseteq (V \times V)$. The set of successors of $v \in V$ is defined by $vE = \{v' \in V | (v, v') \in E\}$.

The games we are interested in are played by two players, called Player 0 and Player 1. We will often fix $\sigma \in \{0, 1\}$ and consider Player σ; if we then want to refer to the other player, we will speak of him or her as Player σ's opponent and write Player $\overline{\sigma}$. Formally, we set $\overline{\sigma} = 1 - \sigma$, for $\sigma \in \{0, 1\}$.

Observe that there is no restriction on the number of the successors of a vertex in an arena.

A play of a game with an arena as above may be imagined in the following way: a token is placed on some initial vertex $v \in V$. If v is a 0-vertex then Player 0 moves the token from v to a successor $v' \in vE$ of v; symmetrically, if v is a 1-vertex then Player 1 moves the token from v to a successor $v' \in vE$ of v. More concisely, when v is a 0-vertex (or 1-vertex), then Player 0 (Player 1) moves the token from v to $v' \in vE$. Next, when v' is a 0-vertex (or 1-vertex), then Player 0 (Player 1) moves the token from v' to $v'' \in v'E$.

This is repeated either infinitely often or until a vertex $\overline{v}$ without successors, a dead end, is reached. Formally, a vertex $\overline{v}$ is called a dead end if $\overline{v} = \varnothing$.

We define a play in the arena A as above as being either

- an infinite path $\pi = v_0 v_1 v_2 \cdots \in V^\omega$ with $v_{i+1} \in v_i E$ for all $i \in \omega$ (infinite play) or
- a finite path $\pi = v_0 v_1 \cdots v_l \in V^+$ with $v_{i+1} \in v_i E$ for all $i < l$, but $v_l E = \varnothing$ (finite play).

A prefix of a play is defined in the obvious way.

Let A be an arena as above and $Win \subseteq V^\omega$. The pair (A, Win) is then called a game G, where A is the arena of the game and Win its winning set. The plays of that game are the plays in the arena A. Player 0 is declared the winner of a play π in the game G iff

- π is a finite play $\pi = v_0 v_1 \cdots v_l \in V^+$ and v_l is a 1-vertex where Player 1 cannot move anymore (when v_l is a dead end) or
- π is an infinite play and $\pi \in Win$.

Player 1 wins π if Player 0 does not win π.

Given an ω-word $\alpha \in \Sigma^\omega$, let

- $Occ(\alpha) = \{a \in \Sigma \mid \exists i, \alpha(i) = a\}$
- $Inf(\alpha) = \{a \in \Sigma \mid \forall i \exists j > i, \alpha(j) = a\}$

where, $Occ(\alpha)$ is the (finite) set of letters occurring in α, and $Inf(\alpha)$ is the (finite) set of letters occurring infinitely often in α.

Let A be as above and assume $\chi : V \longrightarrow C$ is some function mapping the vertices of the arena to a finite set C of so-called colours; such a function will be called a colouring function. The colouring function is extended to plays in a straightforward way. When $\pi = v_o v_1 \cdots .$ is a play, then its colouring, $\chi(\pi)$, is given by $\chi(\pi) = \chi(v_o)\chi(v_1)\chi(v_2)\cdots$. So, when C is viewed as the state set of a finite ω-automaton and Acc is an acceptance condition for this automaton, then $W_\chi(Acc)$ represents the winning set consisting of all infinite plays π where $\chi(\pi)$ is accepted according to Acc. Depending on the actual acceptance condition we are interested in, this means the following, where π stands for any element of V^ω.

- Muller condition $(Acc = F \subseteq P_0(C)) : \pi \in W_\chi(Acc)$ iff $Inf(\chi(\pi)) \in F$.
- Rabin condition $(Acc = \{(E_0, F_0), (E_1, F_1), \ldots, (E_{m-1}, F_{m-1})\})$:
  $\pi \in W_\chi(Acc)$ iff $\exists k \in [m]$, such that $Inf(\chi(\pi)) \cap E_k = \varnothing$ and $Inf(\chi(\pi)) \cap F_k \neq \varnothing$
- Streett condition $(Acc = \{(E_0, F_0), (E_1, F_1), \ldots, (E_{m-1}, F_{m-1})\})$:
  $\pi \in W_\chi(Acc)$ iff $\forall k \in [m].(Inf(\chi(\pi)) \cap E_k \neq \varnothing \vee Inf(\chi(\pi)) \cap F_k = \varnothing)$

- Parity conditions (the colour set C is a finite subset of the integers):
  - max-parity condition: $\pi \in W_\chi(Acc)$ iff $max(Inf(\chi(\pi)))$ is even.
  - min-parity condition: $\pi \in W_\chi(Acc)$ iff $min(Inf(\chi(\pi)))$ is even.
- Büchi condition ($Acc = F \subseteq C$): $\pi \in W_\chi(Acc)$ iff $Inf(\chi(\pi)) \cap F \neq \varnothing$
- 1-winning ($Acc = F \subseteq C$): $\pi \in W_\chi(Acc)$ iff $Occ(\chi(\pi)) \cap F \neq \varnothing$

To indicate that we are working with a certain acceptance/winning condition, we will speak of Muller, Rabin, Streett, Parity, Büchi games.

In order to be able to define formally what it means for a player to win a game, we need to introduce the notion of strategy. Let A be an arena as usual, $\sigma \in \{0, 1\}$, and $f_\sigma : V^*V_\sigma \longrightarrow V$ a partial function. A prefix of a play $\pi = v_0v_1 \dots v_l$ is said to be conform with f_σ if for every i with $0 \leq i < l$ and $v_i \in V_\sigma$ the function f_σ is defined at $v_0 \dots v_i$ and we have $v_{i+1} = f_\sigma(v_0 \dots v_i)$. Note, that this also implies $v_{i+1} \in v_iE$. A play (finite or infinite) is conform with f_σ if each of its prefixes is conform with f_σ. Now we call the function f_σ a strategy for Player σ on $U \subseteq V$ if it is defined for every prefix of a play which is conform with it, starts in a vertex from U, and does not end in a dead end of Player σ. When U is a singleton $\{v\}$, we say f_σ is a strategy for Player σ in v.

Let $G = (A, Win)$ be an arbitrary game with A as usual, and f_σ a strategy for Player σ on U. The strategy f_σ is said to be a winning strategy for Player σ on U if all plays which are conform with f_σ and start in a vertex from U are wins for Player σ.

7.3 Classical Solution with Safra's Determinization

The classical approach for solving the LTL realizability problem is showed by Pnueli and Rosner [1989], that considered the synthesis of a reactive module with input x and output y, which is specified by the linear temporal formula $\varphi(x, y)$; the specification $\varphi(x, y)$ characterizes the expected relation between the input x presented to the program and the output y computed by the program. They show that there exists a program satisfying φ if and only if the branching time formula $(\forall x)(\exists y)A\varphi(x, y)$ is valid over all tree models. For the restricted case that all variables range over finite

domains, the validity problem is decidable, and the authors present an algorithm for constructing the program whenever it exists. The algorithm is based on a procedure for checking the emptiness of Rabin automata on infinite trees in time exponential in the number of pairs, but only polynomial in the number of states. Their procedure leads to a synthesis algorithm whose complexity is double exponential in the length of the given specification.

The two main works on the synthesis of concurrent programs, which are reported by Clarke and Emerson [1981] and by Manna and Wolper [1984], consider a temporal specification φ, and show that if it is satisfiable, the model, that satisfies φ, can be used to construct a program that implements φ. The approach showed in these two pioneering contributions presents some limitations due to the fact that it is based on satisfiability of the formula expressing the specification $\varphi(x, y)$. The implied limitations are that the approach can in principle synthesize only entire or closed systems.

To be clear, we can assume that the system to be constructed has two components, C_1 and C_2. Assume that only C_1 can modify a shared variable x used for communication, and only C_2 can modify y. The fact that $\varphi(x, y)$ is satisfiable means that there exists at least one behavior, listing the running values of x and y, which satisfies $\varphi(x, y)$. This shows that there is a way for C_1 and C_2 to cooperate in order to achieve φ. The hidden assumption is that we have the power to construct both C_1 and C_2 in a way that will ensure the needed cooperation. This is quite satisfactory if we are constructing a closed system consisting solely of C_1 and C_2 and having no additional external interaction.

On the other hand, in a situation typical to an open system, C_1 represents the environment over which the implementor has no control, while C_2 is the body of the system itself, that may be referred as a reactive module. This situation resembles a two-person game. The module C_2 manipulates y, doing its best to maintain $\varphi(x, y)$, despite all the possible x values the environment keeps feeding it. C_1 represents the environment and does its worst to foil the attempts of C_2. The main point is that we have to show that C_2 has a winning strategy for y against all possible x scenarios the environment may present to it.

The natural way to express the existence of a winning strategy for C_2, is again expressed by the AE-formula $(\forall x)(\exists y)\varphi(x, y)$. The only difference is that now it should be interpreted over temporal logic, where x and y are no longer simple variables, but

rather sequences of values assumed by the variables x and y over the computation.

The theorem proving approach to the synthesis of a reactive module should be based on proving the validity of an AE-formula. The precise form of the formula claiming the existence of a program satisfying the linear time temporal formula $\varphi(x, y)$, is $(\forall x)(\exists y)A\varphi(x, y)$, where A is the "for all paths" quantifier of branching time logic. For that reason, although the specification $\varphi(x, y)$ is given in linear logic, which is generally considered adequate for reactive specifications, the synthesis problem has to be solved in a branching framework. This conclusion applies to the synthesis of both synchronous and asynchronous programs.

An interesting observation is that the explicit quantification over the dynamic (i.e., variables that may change their values over the computation) interface variables, x and y, is not absolutely necessary. Indeed, as the authors show in the paper, there exists an equivalent branching time formula, which quantifies only over static variables (i.e., variables which remain constant over the computation), whose satisfiability guarantees the existence of a program for $\varphi(x, y)$.

If we consider the case of finite state programs, this other formula becomes purely propositional, and its satisfiability, therefore, can be resolved by known decision methods for satisfiability of propositional branching time formulae.

For the more general case that deductive techniques have to be applied, Pnueli and Rosner [1989] prefer to establish validity, rather than satisfiability, in particular since the explicitly quantified version emphasizes the asymmetry between the roles of the variables x and y in the program. First, they consider the general case and show that the synthesis formula is valid iff there exists a strategy tree for the process controlling y. Then, the authors argue that such a strategy tree represents a program by specifying an appropriate y for each history of x values. They do not pay attention to the question of how effective this representation of a program is, which becomes relevant when they wish to obtain a program represented in a conventional programming language.

Moreover, they consider the more restricted case in which the specification refers only to Boolean variables. In this case the validity of the synthesis formula is decidable, and they present an algorithm for checking its validity and extracting a finite state program out of a valid synthesis formula.

Another important result of the work of Pnueli and Rosner is a derivation of a better emptiness checking algorithm, whose complexity is deterministic polynomial

time in the number of states and exponential in the number of pairs in the acceptance condition of the automata. Using this improved algorithm, the complete synthesis process can be performed in deterministic time which is doubly exponential in the size of the specification.

7.4 Safraless Approaches for LTL Synthesis

The automata-theoretic approach is one of the most fundamental approaches to developing decision procedures in mathematical logics. To decide whether a formula in a logic with the tree-model property is satisfiable, one constructs an automaton that accepts all (or enough) tree models of the formula and then checks that the language of this automaton is nonempty.

The standard approach translates formulas into alternating parity tree automata, which are then translated, via Safra's determinization construction, into nondeterministic parity automata. This approach is not amenable to implementation because of the difficulty of implementing Safra's construction and the nonemptiness test for nondeterministic parity tree automata. An alternative to the standard automata-theoretic approach is presented by Kupferman and Vardi [2005b], that avoid the Safra's construction and nondeterministic parity tree automata using universal co-Büchi tree automata and nondeterministic Büchi tree automata. While their translations have the same complexity as the standard approach, they are significantly simpler, less difficult to implement, and have practical advantages like being amenable to optimizations and a symbolic implementation.

On the other hand, Filiot et al. [2009], present a reduction from the LTL realizability problem to a game with an observer that checks that the game visits a bounded number of times accepting states of a universal co-Büchi word automaton. The authors show that such an observer can be made deterministic and that this deterministic observer has a nice structure which can be exploited by an incremental algorithm that manipulates antichains of game positions.

7.4.1 A Rank construction

One of the most fundamental approaches to developing decision procedures in mathematical logics is the automata-theoretic approach [Rabin, 1969].

It is based on the fact that many logics enjoy the tree-model property so if a formula in the logic is satisfiable then it has a tree (or a tree-like) model [Vardi, 1997].

To decide whether a formula in such a logic is satisfiable, it can be constructed an automaton A that accepts all (or enough) tree models of the formula and then checks that the language of A is nonempty. The automata-theoretic approach was developed first for monadic logics over finite words [Büchi, 1960; Elgot, 1961; Trakhtenbrot, 1962].

Then, it was extended to infinite words by Büchi [1962], to finite trees by Thatcher and Wright [1968], and finally generalized to infinite trees by Rabin [1969].

If we consider the Rabins fundamental result, we use SnS, the monadic theory of infinite trees, to show decidability of a logic, simply demonstrating an effective reduction of that logic to SnS [Gabbay, 1972; Kozen and Parikh, 1983].

The negative aspect is that the complexity of SnS is known to be nonelementary [Meyer, 1975].

For that reason, in the early 1980s, when decidability of highly expressive logics became of practical interest in areas such as formal verification and Artificial Intelligence [De Giacomo and Lenzerini, 1994; Kozen, 1983], and complexity-theoretic considerations started to play a greater role, the original automata-theoretic idea was revived. By the mid 1980s, the focus was on using automata to obtain tighter upper bounds. This required progress in the underlying automata-theoretic techniques. Thus, that progress was attained by Safra [1988], who described an optimal determinization construction for automata on infinite words, and by Emerson and Jutla [1988], and Pnueli and Rosner [1989], who described improved algorithms for parity tree automata.

The introduction of alternating automata on infinite trees gave further simplification [Emerson and Jutla, 1991; Muller and Schupp, 1987].

The standard approach for checking whether a formula ψ is satisfiable, is composed by three steps: first, construct an alternating parity tree automaton A that accepts all (or enough) tree models of ψ; then translate this automaton to a nondeterministic parity

tree automaton B and, finally, check that the language of B is nonempty.

While this standard approach yielded significantly improved upper bounds (in some cases reducing the upper time bound from octuply exponential [Streett, 1982] to singly exponential [Vardi, 1998]), it proved to be not too amenable to implementation. Indeed, in the second step, removing alternation from alternating tree automata involves determinization of word automata, and Safras construction proved quite resistant to efficient implementation [Tasiran et al., 1995].

An alternative removal of alternation is described by Muller and Schupp [1995].

Like Safra's construction, however, this translation is very complicated [Althoff et al., 2005].

Moreover, the best-known algorithms for parity-tree-automata emptiness are exponential [Jurdzinski, 2000].

Thus, while highly optimized software packages for automata on finite words and finite trees have been developed over the last few years [Elgaard et al., 1998], no such software has been developed for automata on infinite trees.

In their work, Kupferman and Vardi [2005b] offer an alternative to the standard automata-theoretic approach, avoiding the use of Safra's construction and of nondeterministic parity tree automata.

In their approach, in order to check whether a formula Ψ is satisfiable one follows these steps: (1) construct an alternating parity tree automaton A that accepts all (or enough) tree models of Ψ, (2) reduce A to a universal co-Büchi automaton B, (3) reduce B to an alternating weak tree automaton C , (4) translate C to a nondeterministic Büchi tree automaton D , and (5) check that the language of D is nonempty.

The central point is avoiding Safra's construction, by using universal co-Büchi automata instead of deterministic parity automata. The positive aspect is that universal automata have the desired property, enjoyed also by deterministic automata but not by nondeterministic automata, of having the ability to run over all branches of an input tree. In addition, the co-Büchi acceptance condition is much simpler than the parity condition. For that reason the nonemptiness problem for universal co-Büchi tree automata can be solved by reducing them into nondeterministic Büchi tree automata. This reduction goes through alternating weak tree automata [Muller et al., 1988], so there is not the need for the parity acceptance condition. The nonemptiness problem for nondeterministic Büchi tree automata is much simpler than the nonemptiness problem

for nondeterministic parity tree automata and it can be solved symbolically and in quadratic time [Vardi and Wolper, 1986].

Universal co-Büchi tree automata are a special case of alternating parity tree automata: the transition function of a universal co-Büchi tree automata contains only conjunctions and the acceptance condition corresponds to a parity condition of index 2. Universal co-Büchi tree automata are indeed strictly less expressive than alternating parity tree automata, but, despite that, they are very powerful.

From one hand, the emptiness problem for alternating parity tree automata is easily reducible to the emptiness problem for universal co-Büchi tree automata. Moreover, it is easy to translate universal co-Büchi tree automata into nondeterministic Büchi tree automata so that emptiness is preserved, that is the nondeterministic Büchi tree automata is empty iff the universal co-Büchi tree automata is empty. Thus, as discussed previously, traditional decidability algorithms that end up in a complicated alternating parity tree automata nonemptiness check, can be much simplified.

Furthemore, universal co-Büchi tree automata are useful for tasks traditionally assigned to alternating parity tree automata. Thus, in some cases, in particular the realizability and synthesis problems for LTL specifications [Pnueli and Rosner, 1989], it is possible to skip the construction of an alternating parity automaton and go directly to a universal co-Büchi automaton.

While this Safraless approach simplifies the algorithms and improves the complexity of the decidability problems, the fact it uses a simplified class of automata (that is, co-Büchi rather than parity) causes the constructions to have more states than these constructed by the traditional algorithm. Safra's determinization construction involves complicated data structures because each state in the deterministic automaton is associated with a labeled ordered tree. Moreover, there is no symbolic implementation of decision procedures that are based on Safra's determinization and nondeterministic parity tree automata.

The translations and reductions, showed by Kupferman and Vardi [2005b], are significantly simpler than the standard approach, making them less difficult to implement, both explicitly and symbolically.

These advantages are obtained with no increase in the complexity. In fact their construction is amenable to several optimization techniques.

In their work, [Kupferman and Vardi, 2005b] solved the problem of a Safraless

decision procedure and showed how universal co-Büchi automata can be used in order to circumvent Safra's determinization and the parity acceptance condition. Their construction avoids the complicated determinization construction of Safra, but its correctness proof makes use of the bounded-size run graph property, which in turn makes use of Safras determinization.

7.4.2 An antichain algorithm

Filiot et al. [2009] introduce a novel Safraless approach to LTL realizability and synthesis, based on universal K-Co-Büchi word automata.

The realizability problem for an LTL formula ϕ is seen as a game between two players [Pnueli and Rosner, 1989].

Each of the players is controlling a subset of the set P of propositions on which the LTL formula ϕ is constructed. The set of propositions P is partitioned into the set of input signals I, that are controlled by *Player input* (the environment, also called Player I), and O the set of output signals that are controlled by *Player output* (the controller, also called Player O). The realizability game is played in turns. Player O is the protagonist, she wants to satisfy the formula ϕ, while Player I is the antagonist as he wants to falsify the formula ϕ.

Player O starts by giving a subset o_0 of propositions, Player I responds by giving a subset of propositions i_0, then Player O gives o_1 and Player I responds by i_1, and so on. This game lasts forever and the outcome of the game is the infinite word $\omega = (i_0 \cup o_0)(i_1 \cup o_1)(i_2 \cup o_2) \cdots \in (2^P)^\omega$. We say that Player O *wins* if the resulting infinite word w is a model of ϕ. This problem is central when dealing with specifications for reactive systems and, in that context, the signals of the environment being uncontrollable, unrealizable specifications are useless as they can not be implemented.

The classical automata-based solution to LTL synthesis, seen in the previous section, can be summarized as follows .

Given an LTL formula ϕ, we construct a nondeterministic Büchi automaton A that accepts all models of ϕ, then we transform A into a deterministic Rabin automaton B using Safra's determinization procedure [Safra, 1988], and use B as an observer in a turn-based two-player game. Unfortunately, this theoretically elegant procedure has turn out to be very difficult to implement. Indeed, Safra's determinization procedure

generates very complex state spaces: states are colored trees of subsets of states of the original automaton and there is not a symbolic data-structure to handle such state spaces.

Moreover, the game to solve as the last step (on a potentially doubly-exponential state-space) is a Rabin game, and this problem is known to be NP-complete.

Recently, Kupferman and Vardi [2005b] have proposed procedures that avoid the determinization step and so the Safra's construction, as we have seen in Section 7.4.1. In particular, they reduce the LTL realizability problem to the emptiness of a Universal Co-Büchi Tree automaton. They show how to test emptiness of a Universal Co-Büchi Tree automaton by translation to an alternating weak Büchi tree automaton, again translated into a non-deterministic Büchi tree automaton for which testing emptiness is easy. All these steps have been implemented and optimized in several ways by Jobstmann and Bloem [2006] in a tool called Lily.

A different and more direct Safraless decision procedure for the LTL realizability and synthesis problem has been proposed in the work of Filiot et al. [2009], where the authors identify structural properties that allow to define an antichain algorithm.

Their procedure uses Universal Co-Büchi Word automaton (UCW). A nice property of those automata is the following: if a Moore machine M with m states defines a language included into the language defined by a UCW with n states, then obviously every run on the words generated by M contains at most $2mn$ accepting states. Consequently, a Moore machine that enforces a language defined by a UCW also enforces a stronger requirement defined by the same automaton where the acceptance condition is strengthened to a so called $2mn$-bounded one: a run is accepting if it passes at most $2mn$ times by an accepting state.

The authors use the Safra's result to obtain that the size of a Moore machine, realizing a language defined by a UCW, can be bounded. This step allow them to give a reduction from the general problem to the problem of the realizability of a k-bounded UCW specification. Contrarily to general UCW specifications, k-bounded UCW specifications can easily be made deterministic and, most importantly, the underlying deterministic automaton is always equipped with a partial-order on states that can be used to efficiently manipulate its state space using their antichain method. They show how to reduce the realizability problem to a safety game. Symbolic algorithms for solving safety games are constructed using the so-called *controllable predecessor operator*, as

it can be seen in [Grädel et al., 2002].

Moreover, Raskin et al. have implemented this new antichain algorithm in a tool called *Acacia*. Acacia is a prototype implementation of their antichain algorithm for LTL realizability and synthesis. Given an LTL formula and a partition of its propositions into inputs and outputs, Acacia tests realizability of the formula: if it is realizable, it outputs a Moore machine representing a winning strategy for the output player, otherwise it outputs a winning strategy for the input player. The experiments show that the antichain algorithm is a very promising approach to LTL synthesis. Although the formulas are still rather small, the results validate the relevance of the method. Indeed, without any further optimization, the results outperform Lily [Kupferman and Vardi, 2005b].

The approach used by Filiot et al. [2009] to described a novel Safraless procedure to LTL realizability and synthesis, differs from the Kupferman and Vardi [2005b] approach in the following points. First, Kupferman and Vardi reduce the realizability problem to a game with a Büchi objective, while their approach reduces it to a game with a safety objective. The second aspect is that the approach of Raskin et al. allows to define a natural partial order on states that can be exploited by an antichain algorithm, which is not obvious in the approach of Kupferman and Vardi.

Finally, in the work of Kupferman and Vardi [2005b], states of alternating weak tree automata are equipped with unique ranks that partition the set of states into layers. States which share the same rank are either all accepting or all non-accepting, indeed the transition function allows one to stay in the same layer or to go in a layer with lower rank. A run is accepting if it gets stuck in a non-accepting layer.

While the notion of counters, used by Raskin et al., looks similar to ranks, defined by Kupferman and Vardi, it is different. Indeed, the notion of rank does not constraint the runs to visit accepting states a bounded number of times, for example bounded by a constant. This is why a Büchi acceptance condition is needed for Kupferman and Vardi, while counting the number of visited accepting states allows Raskin et al. to define a safety acceptance condition.

7.5 The Realizability Problem for Timed Specification

In a recent work [Di Giampaolo et al., 2010a], we generalize the ideas of Filiot et al. [2009] to solve the realizability problem for timed specification and, in particular, for a fragment of the Event Clocks Logic called $\mathsf{LTL}_{\triangleleft}$ [Doyen et al., 2009]. For each formula of this logic, we can construct, in exponential time, a universal event-clock automaton with co-Büchi acceptance condition that accepts the set of timed words that the formula defines. Then, we show that the co-Büchi acceptance condition can be strengthened into a condition that asks that all runs of the automaton visit less than $K \in \mathbb{N}$ times the set of accepting locations. This allows to reduce the realizability problem for $\mathsf{LTL}_{\triangleleft}$ to the realizability problem for universal K-co-Büchi event-clock automata. Those are easily determinizable and this reduces the original problem to a timed safety game problem. We will show, in Section 7.7.3, that this timed safety game problem can be solved using the tool UPPAAL TIGA [Behrmann et al., 2007].

We start by giving the definitions of *timed game*, *region game* and *parity game*.

7.5.1 Timed games

We concentrate our attention to the *realizability problem* for timed specifications expressed by UECA. We restrict to event-clock automata with history clocks only as the use of prophecy clocks leads to undecidability [Doyen et al., 2009]. To formalize the *realizability problem* in this context, we rely on the notion of *timed game*. A *timed game* (TG for short) is a tuple $\langle \Sigma_1, \Sigma_2, W \rangle$ where Σ_i $(i = 1, 2)$ is a finite alphabet for player i (with $\Sigma_1 \cap \Sigma_2 = \varnothing$), and $W \subseteq \mathsf{T}\Sigma^\omega$ is a set of timed words, called the *objective* of the game (for player 1).

A TG is played for infinitely many rounds. At each round i, player 1 first chooses a delay t_i^1 and a letter $\sigma_i^1 \in \Sigma_1$. Then, player 2 chooses either to pass or to overtake player 1 with a delay $t_i^2 \leq t_i^1$ and a letter $\sigma_i^2 \in \Sigma_2$. A *play* in a timed game is a timed word (w, τ) s.t. for any $i \geq 0$ either (i) player 2 has passed at round i, $w_i = \sigma_i^1$ and $\tau_i = \tau_{i-1} + t_i^1$, or (ii) player 2 has overtaken player 1 at round i, $w_i = \sigma_i^2$ and $\tau_i = \tau_{i-1} + t_i^2$ (with the convention that $\tau_{-1} = 0$). A timed word θ is *winning* in $\langle \Sigma_1, \Sigma_2, W \rangle$ iff $\theta \in W$. A *strategy* for player 1 is a function π that associates to every finite prefix of a timed word $(w_0, \tau_0) \dots (w_k, \tau_k)$ an element from $\Sigma_1 \times \mathbb{R}^{\geq 0}$. A play $\theta = (w, \tau)$ is

consistent with strategy π for player 1 iff for every $i \geq 0$, *either* player 1 has played according to its strategy i.e., $(w_i, \tau_i - \tau_{i-1}) = \pi((w_0, \tau_0) \dots (w_{i-1}, \tau_{i-1}))$ *or* player 2 has overtaken the strategy of player 1 i.e., $w_i \in \Sigma_2$, and $\pi((w_0, \tau_0) \dots (w_{i-1}, \tau_{i-1})) = (\sigma, \tau)$ with $\tau \geq \tau_i - \tau_{i-1}$. The *outcome* of a strategy π in a game $G = \langle \Sigma_1, \Sigma_2, W \rangle$, noted $\mathsf{Outcome}\,(G, \pi)$ is the set of all plays of G that are consistent with π. A strategy π is *winning* iff $\mathsf{Outcome}\,(G, \pi) \subseteq W$.

The *realizability problem* asks, given a universal PastECA A with co-Büchi acceptance condition, whose alphabet Σ is partitioned into Σ_1 and Σ_2, if player 1 has a winning strategy in $G = \langle \Sigma_1, \Sigma_2, \mathsf{L}_{\mathsf{coB}}(A) \rangle$.

To solve this problem without using Safra determinization, we show how to reduce it to a timed safety objective via a strengthening of the winning objective using K-co-Büchi acceptance condition.

7.5.2 Region games

A *region game* is a tuple $G^R = \langle \Sigma_1, \Sigma_2, cmax, W \rangle$ where $\Sigma = \Sigma_1 \cup \Sigma_2$, $\Sigma_1 \cap \Sigma_2 = \varnothing$, $cmax \in \mathbb{N}$, W is a set of infinite words on the alphabet $(\Sigma_1 \uplus \Sigma_2) \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)$, called the *objective* of the game (for player 1).

A *play* of a region game is an infinite (untimed) word on the alphabet $(\Sigma_1 \cup \Sigma_2) \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)$. The game is played for infinitely many rounds. In the initial round, player 1 first chooses a letter $\sigma^1 \in \Sigma_1$. Then, either player 2 lets player 1 play and the first letter of the play is $(\sigma^1, r_{\text{in}})$, or player 2 overtakes player 1 with a letter $\sigma^2 \in \Sigma_2$ and the first letter of the play is $(\sigma^2, r_{\text{in}})$. In all the subsequent rounds, and assuming that the prefix of the current play is $(\sigma_0, r_0), \cdots (\sigma_k, r_k)$, player 1 first chooses a pair $(\sigma^1, r^1) \in \Sigma_1 \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)$ such that $r_k[\overleftarrow{x_{\sigma_k}} := 0] \leq_{\mathsf{t.s.}} r^1$. Then, either player 2 lets player 1 play and the new prefix of the play is $\rho_{k+1} = \rho_k \cdot (\sigma^1, r^1)$, or player 2 decides to overtake player 1 with a pair $(\sigma^2, r^2) \in \Sigma_2 \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)$, respecting $r_k[\overleftarrow{x_{\sigma_k}} := 0] \leq_{\mathsf{t.s.}} r^2 \leq_{\mathsf{t.s.}} r^1$. In this case, the new prefix of the play is $\rho_{k+1} = \rho_k \cdot (\sigma^2, r^2)$. A play ρ is *winning* in $\langle \Sigma_1, \Sigma_2, cmax, W \rangle$ iff $\rho \in W$. As for timed games, a *strategy* for player 1 is a function π^R that associates to every finite prefix $(w_0, r_0) \dots (w_k, r_k)$ an element $(\sigma, r) \in \Sigma_1 \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)$ such that $r_k[\overleftarrow{x_{w_k}} := 0] \leq_{\mathsf{t.s.}} r$. A play $\rho = (\sigma_0, r_{\text{in}})(\sigma_1, r_1) \cdots$ is consistent with strategy π for player 1 iff for all $i \geq 0$, *either* player 1 has played according to its strategy, i.e., $(\sigma_i, r_i) =$

$\pi((\sigma_0, r_{\text{in}}) \dots (\sigma_{i-1}, r_{i-1}))$ (with the convention that $(\sigma_{-1}, r_{-1}) = \varepsilon$), *or* player 2 has overtaken the strategy of player 1 i.e., $\sigma_i \in \Sigma_2$, $\pi((\sigma_0, r_{\text{in}}) \dots (\sigma_{i-1}, r_{i-1})) = (\sigma, r)$ and $r_i \leq_{\text{t.s.}} r$.

Remark 49 *All plays of* $\langle \Sigma_1, \Sigma_2, cmax, W \rangle$ *are in* $\mathsf{L_B}(\mathsf{RegAut}\,(\mathbb{H}_\Sigma, cmax))$.

The *outcome* $\mathsf{Outcome}\,(G, \pi)$ of a strategy π on a region game G and winning strategies are defined as usual.

7.5.3 Parity games

A *parity game* is a tuple $G = \langle Q, E, q_0, \mathsf{Colours}, \lambda \rangle$ where $Q = Q_1 \uplus Q_2$ is the set of *positions*, partitioned into the player 1 and player 2 positions, $E \subseteq Q \times Q$ is the set of *edges*, $q_0 \in Q$ is the initial position, and $\lambda : Q \mapsto \mathsf{Colours}$ is the *coloring function*.

A *play* of a parity game $G = \langle Q, E, q_0, \mathsf{Colours}, \lambda \rangle$ is an infinite sequence $\rho = q_0 q_1 \cdots q_j \cdots$ of positions s.t. for any $j \geq 0$: $(q_j, q_{j+1}) \in E$. Given a play $\rho = q_0 q_1 \cdots q_j \cdots$, we denote by $\mathsf{Inf}\,(\rho)$ the set of positions that appear infinitely often in ρ, and by $\mathsf{Par}\,(\rho)$ the value $\max\{\lambda(q) \mid q \in \mathsf{Inf}\,(\rho)\}$. A play ρ is *winning* for player 1 iff $\mathsf{Par}\,(\rho)$ is *even*. A *strategy* for player 1 in G is a function $\pi : Q^* Q_1 \to Q$ that associates, to each finite prefix ρ of play ending in a Player 1 state $\mathsf{Last}(\rho)$, a successor position $\pi(\rho)$ s.t. $(\mathsf{Last}(\rho), \pi(\rho)) \in E$. Given a parity game G and a strategy π for player 1, we say that a play $\rho = q_0 q_1 \cdots q_j \cdots$ of G *is consistent with* π iff for $j \geq 0$: $q_j \in Q_1$ implies that $q_{j+1} = \pi(q_0 \cdots q_j)$. We denote by $\mathsf{Outcome}\,(G, \pi)$ the set of plays that are consistent with π. A strategy π is *winning* iff every play $\rho \in \mathsf{Outcome}\,(G, \pi)$ is winning.

It is well-known that parity games admit *memoryless strategies*. More precisely, if there exists a winning strategy for player 1 in a parity game G, then there exists a winning strategy π for player 1 s.t. for any pair of prefixes ρ and ρ': $\mathsf{Last}(\rho) = \mathsf{Last}(\rho')$ implies $\pi(\rho) = \pi(\rho')$. A memoryless strategy π can thus be finitely represented by a function $f_\pi : Q_1 \to Q$, where, for any $q \in Q_1$, $f_\pi(q)$ is the (unique) position q' s.t. for any prefix $\rho = q_0 \cdots q$, $\pi(\rho) = q'$. In the sequel we often abuse notations and confuse f_π with π when dealing with memoryless strategies in parity games.

7.6 Reduction to Timed Safety Game

In this section we focus on the reduction from the original problem to the timed safety game problem. The main result of this section is the following theorem:

Theorem 50 *Given a universal* PastECA A *with co-Büchi acceptance condition, whose alphabet* Σ *is partitioned into* Σ_1 *and* Σ_2*, player 1 has a winning strategy in* $G^T = \langle \Sigma_1, \Sigma_2, \mathsf{L}_{\mathsf{coB}}(A) \rangle$ *iff he has a winning strategy in* $G^T_K = \langle \Sigma_1, \Sigma_2, \mathsf{L}_{\mathsf{KcoB}}(A) \rangle$*, for any* $K \geq (2n^{n+1}n! + n) \times |\mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)\,|$ *where* n *is the number of locations in* A*.*

To establish this result, we use several intermediary steps. First, we show that we can associate a game with an ω-regular objective, played on untimed words, to any timed game whose objective is defined by a UECA with co-Büchi acceptance condition.

The next proposition shows how a timed game can be reduced to a region game.

Proposition 51 *Let* A *be a universal* PastECA *with maximal constant* $cmax$*. Player 1 has a winning strategy in the timed game* $G^T = \langle \Sigma_1, \Sigma_2, \mathsf{L}_{\mathsf{coB}}(A) \rangle$ *iff he has a winning strategy in the region game* $G^R = \langle \Sigma_1, \Sigma_2, cmax, \mathsf{L}_{\mathsf{coB}}(\mathsf{Rg}(A)) \rangle$*. Moreover, for any* $K \in \mathbb{N}$*, player 1 has a winning strategy in* $G^T = \langle \Sigma_1, \Sigma_2, \mathsf{L}_{\mathsf{KcoB}}(A) \rangle$ *iff he has a winning strategy in* $G^R = \langle \Sigma_1, \Sigma_2, cmax, \mathsf{L}_{\mathsf{KcoB}}(\mathsf{Rg}(A)) \rangle$*.*

Proposition 51 tells us that we can reduce the realizability problem of timed games to that of *region games*. Next we show that *region games* can be won thanks to a *finite memory strategy*. For that, we expose a reduction from region games to *parity games*.

Let us show how to reduce the region game $\langle \Sigma_1, \Sigma_2, cmax, \mathsf{L}_{\mathsf{coB}}(\mathsf{Rg}(A)) \rangle$ to a parity game. First consider the NWA $\widetilde{\mathsf{Rg}(A)}$ that dualizes $\mathsf{Rg}(A)$ and such that $\mathsf{L}_{\mathsf{B}}(\widetilde{\mathsf{Rg}(A)}) = \Sigma^\omega \setminus \mathsf{L}_{\mathsf{coB}}(\mathsf{Rg}(A))$. Then, using Piterman [2007] construction , we can obtain a deterministic parity automaton $\widetilde{D}$ such that $\mathsf{L}_{\mathsf{P}}(\widetilde{D}) = \mathsf{L}_{\mathsf{B}}(\widetilde{\mathsf{Rg}(A)})$, and by complementing $\widetilde{D}$, we obtain a deterministic (and complete) parity automaton D such that $\mathsf{L}_{\mathsf{P}}(D) = \mathsf{L}_{\mathsf{coB}}(\mathsf{Rg}(A))$. We use this automaton and the region automaton $\mathsf{RegAut}\,(\mathbb{H}_\Sigma, cmax)$ as a basis for the construction of the parity game.

A play in the parity game simulates runs over words in $(\Sigma \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax))^\omega$ of both $D = \left\langle Q^D, q^D_{in}, \Sigma \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)\,, \delta^D, \alpha^D \right\rangle$ and $\mathsf{RegAut}\,(\mathbb{H}_\Sigma, cmax) = \left\langle Q^R, q^R_{in}, \Sigma^R, \delta^R, \alpha^R \right\rangle$. Formally, $G_D = \left\langle q^G_{in}, Q^G, E^G, \mathsf{Colours}, \lambda^G \right\rangle$, where the positions of player 1 are $Q^G_1 = (Q^D \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax))$, and the positions of player 2

are $Q_2^G = (Q^D \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)) \times (\Sigma_1 \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax))$. Intuitively, $(q, r) \in Q_1^G$ means that the simulated runs are currently in the states q and r of respectively D and $\mathsf{RegAut}\,(\mathbb{H}_\Sigma, cmax)$. From a position in Q_1^G, player 1 can go to a position memorizing the current states in D and $\mathsf{RegAut}\,(\mathbb{H}_\Sigma, cmax)$, as well as the next move according to player 1's strategy. Thus, $(q, r, \sigma^1, r^1) \in Q_2^G$ means that we are in the states q and r in the automata, and that (σ^1, r^1) is the letter proposed by player 1. Then, from (q, r, σ^1, r^1), player 2 chooses either to let player 1 play, or decides to overtake him. In the former case, the game moves a position (q', r') where q' and r' are the new states in D and $\mathsf{RegAut}\,(\mathbb{H}_\Sigma, cmax)$ after a transition on (σ^1, r^1). In the latter case (overtake player 1), the game moves to a position (q'', r''), assuming there are $\sigma^2 \in \Sigma_2$, $r^2 \leq_{\text{t.s.}} r^1$ such that q'' and r'' are the new states of D and $\mathsf{RegAut}\,(\mathbb{H}_\Sigma, cmax)$ after a transition on (σ^2, r^2). These moves are formalized by the set of edges $E^G = E_1^G \uplus E_2^G$ where:

$$
\begin{aligned}
E_1^G &= \{\big((q, r), (q, r, \sigma^1, r^1)\big) \mid \sigma^1 \in \Sigma_1, \delta^R(r, (\sigma^1, r^1)) \neq \bot\} \\
E_2^G &= \{\big((q, r, \sigma^1, r^1), (q', r')\big) \mid (q', r') = (\delta^D(q, (\sigma^1, r^1)), \delta^R(r, (\sigma^1, r^1))\} \\
&\quad \cup \left\{ \big((q, r, \sigma^1, r^1), (q', r')\big) \;\middle|\; \begin{array}{c} \exists \sigma^2 \in \Sigma_2, r^2 \leq_{\text{t.s.}} r^1, \delta^R(r, (\sigma^2, r^2)) \neq \bot, and \\ (q', r') = (\delta^D(q, (\sigma^2, r^2)), \delta^R(r, (\sigma^2, r^2))) \end{array} \right\}
\end{aligned}
$$

Intuitively, player 1 chooses its next letter in Σ_1 and a region. The definition of E_1^G uses transitions of $\mathsf{RegAut}\,(\mathbb{H}_\Sigma, cmax)$ and hence enforces the fact that player 1 can only propose to go to a region that is a time successor of the current region, and thus respects the rules of the region game. Symmetrically, player 2 can either let player 1 play, or play a letter from Σ_2 with a region which is a time predecessor of the region proposed by player 1. Again, the automaton D being complete, player 2 can play any letter in Σ_2, but he can only play in regions that are time successors of the current region. The initial position is $q_{in}^G = (q_{in}^D, r_{\text{in}})$. Finally, the labelling of the positions reflects the colouring of the states in D: $\lambda^G(q, r) = \lambda^G(q, r, \sigma^1, r^1) = \alpha^D(q)$. Hence, a play in the parity game is winning for player 1 if and only if the word simulated is accepted by D. The next proposition shows the relationship between G_R and the corresponding parity game G_D.

Proposition 52 *Player 1 has a winning strategy in the region game G_R* if and only if *he has a winning strategy in the corresponding parity game G_D.*

Because parity games admit memoryless strategies, and thanks to Proposition 52, we can deduce a bound on the memory needed to win a region game whose objective is given by $\mathsf{L}_{\mathsf{coB}}(\mathsf{Rg}(A))$ for a universal PastECA A.

Lemma 53 *Let A be a universal* PastECA *with n locations and maximal constant $cmax$. If player 1 has a winning strategy in $G_R = \langle \Sigma_1, \Sigma_2, cmax, \mathsf{L}_{\mathsf{coB}}(\mathsf{Rg}(A))\rangle$, then he has a finite-state strategy, represented by a deterministic finite state transition system with at most m states, where $m = (2n^n n! + 1) \times |\mathsf{Reg}\left(\mathbb{H}_\Sigma, cmax\right)|$.*

Proof If player 1 has a winning strategy in $\langle \Sigma_1, \Sigma_2, cmax, \mathsf{L}_{\mathsf{coB}}(\mathsf{Rg}(A))\rangle$, then by Proposition 52, and by the memoryless property of parity games, he has a memoryless winning strategy in the parity game G_D, $\pi^G : Q_1^G \to Q_2^G$. From this memoryless strategy, one can define a finite-state strategy for player 1 in the original region game. We first define $\pi : Q_1^G \to \Sigma_1 \times \mathsf{Reg}\left(\mathbb{H}_\Sigma, cmax\right)$ as follows. For all $q \in Q^D$, $r \in \mathsf{Reg}\left(\mathbb{H}_\Sigma, cmax\right)$: $\pi(q, r) = (\sigma^1, r^1)$ iff $\pi^G(q, r) = (q, r, \sigma^1, r^1)$. Then, we let A^π be the finite transition system $\left\langle Q_1^G, q_{in}^G, \Sigma \times \mathsf{Reg}\left(\mathbb{H}_\Sigma, cmax\right), \delta^\pi \right\rangle$ where, for all $q = (q_1, r_1) \in Q_1^G$, $(\sigma, r) \in \Sigma \times \mathsf{Reg}\left(\mathbb{H}_\Sigma, cmax\right)$: $\delta^\pi(q, (\sigma, r)) = (q_1', r_1')$ iff (i) $q_1' = \delta^D(q_1, (\sigma, r))$ and (ii) $r_1' = \delta^R(r_1, (\sigma, r)))$ and (iii) either $\pi(q) = (\sigma, r)$, or $\pi(q) = (\sigma', r')$ and $\sigma \in \Sigma_2$, and $r \leq_{\mathsf{t.s.}} r'$. In the other cases, δ is undefined.

From π and A^π, we can define the strategy π^R to be played in the region game as follows. Let $\Delta : Q_1^G \times (\Sigma \times \mathsf{Reg}\left(\mathbb{H}_\Sigma, cmax\right))^* \to Q_1^G \cup \{\bot\}$ be the function s.t. $\Delta(q, w)$ is the location reached in A^π after reading the finite word w from location q, or $\bot$ if w cannot be read from q. Then, π^R is defined as follows. For any $\rho^R = (\sigma_1, r_1) \cdots (\sigma_n, r_n)$, we let $\pi^R(\rho^R) = \pi(\Delta(q_{in}^G, \rho^R))$ if $\Delta(q_{in}^G, \rho^R) \neq \bot$; otherwise: $\pi^R(\rho^R) = (\sigma, r_n)$ where σ is any letter in Σ_1. Remark that, by definition of π and A^π, the proposed region is always a time successor of the last region of the play, so the strategy is correctly defined. Let us show that π^R is winning: let $\rho^R = (\sigma_0, r_0) \cdots (\sigma_j, r_j) \cdots$ be a play consistent with π^R. By definition of π^R, there is a run $R = q_{in}^G, (\sigma_0, r_0), q_1^G, \cdots, q_j^G, (\sigma_j, r_j), \cdots$ of A^π over ρ^R. It is easy to see that one can construct from this run a play in G^D that is consistent with π^G. Then, since π^G is winning, $\rho^R \in \mathsf{L}_{\mathsf{P}}(D) = \mathsf{L}_{\mathsf{coB}}(\mathsf{Rg}(A))$. Since this is true for any run that is consistent with π^R, it is a winning strategy.

Finally, observe that the number of states of A^π is $|Q^D| \times |\mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)\,|$. By the result of [Piterman, 2007], $|Q^{\tilde{D}}| = 2n^n n!$. Then $|Q^D| = 2n^n n! + 1$, and this establishes the bound $m = (2n^n n! + 1) \times |\mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)\,|$. □

Thanks to Lemma 53, we can now prove that we can strengthen the co-Büchi condition of the objective of the region game, to a K-co-Büchi condition:

Proposition 54 *Let A be a universal* PastECA *with co-Büchi acceptance condition, n locations and maximal constant $cmax$. Then, player 1 has a winning strategy in $G^R = \langle \Sigma_1, \Sigma_2, cmax, \mathsf{L}_{\mathsf{coB}}(\mathsf{Rg}(A))\rangle$* if and only if *he has a winning strategy in $G^R_K = \langle \Sigma_1, \Sigma_2, cmax, \mathsf{L}_{\mathsf{KcoB}}(\mathsf{Rg}(A))\rangle$, with $K = (2n^{n+1}n! + n) \times |\mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)\,|$.*

Proof First, observe that, since $\mathsf{L}_{\mathsf{KcoB}}(\mathsf{Rg}(A)) \subseteq \mathsf{L}_{\mathsf{coB}}(\mathsf{Rg}(A))$, any winning strategy for player 1 in G^R_K, is winning in G^R.

Conversely, suppose player 1 has a winning strategy in G^R. Then, by Lemma 53, there is a strategy π and a transition system $A^\pi = \langle Q^\pi, q^\pi_{in}, \Sigma \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)\,, \delta^\pi\rangle$ with m locations (where $m = (2n^n n!+1)\times|\mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)\,|$) s.t. $\mathsf{Outcome}\,(G_R, \pi) = \mathsf{L}(A^\pi)$ and $\mathsf{L}(A^\pi) \subseteq \mathsf{L}_{\mathsf{coB}}(\mathsf{Rg}(A))$. Let $\mathsf{Rg}(A) = \langle Q, q_{in}, \Sigma \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)\,, \delta, \alpha\rangle$, and let $A^\pi \times \mathsf{Rg}(A) = \langle Q^\pi \times Q, (q^\pi_{in}, q_{in}), \Sigma \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)\,, \delta'\rangle$ be the transition system s.t. for all $(q^\pi, q) \in Q^\pi \times Q$, for all $(\sigma, r) \in \Sigma \times \mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)$: $(q^\pi_2, q_2) \in \delta'((q^\pi_1, q_1), (\sigma, r))$ iff $\delta^\pi(q^\pi_1, (\sigma, r)) = q^\pi_2$ and q_2 appears as a conjunct in $\delta(q_1, (\sigma, r))$ (recall that $\mathsf{Rg}(A)$ is universal). Clearly, each run of $A^\pi \times \mathsf{Rg}(A)$ simulates a run of A^π, together with a branch that has to appear in a run of $\mathsf{Rg}(A)$.

Then, let us show that there is, in $A^\pi \times \mathsf{Rg}(A)$, no cycle that contains a location from $Q^\pi \times \alpha$. This is established by contradiction. Assume such a cycle exists, and let $(q^\pi_{in}, q_{in})(q^\pi_1, q_1)(q^\pi_2, q_2)\cdots(q^\pi_j, q_j)\cdots$ be an infinite run of $A^\pi \times \mathsf{Rg}(A)$ that visits a location from $Q^\pi \times \alpha$ infinitely often. Moreover, let w be the infinite word labeling this run. Then, clearly, $q^\pi_{in} q^\pi_1 q^\pi_2 \cdots q^\pi_j \cdots$ is a run of A^π that accepts w. On the other hand, the run of $\mathsf{Rg}(A)$ on w necessarily contains a branch labelled by $q_{in} q_1 q_2 \cdots q_j \cdots$. Since this branch visits α infinitely often, $\mathsf{Rg}(A)$ rejects w because the acceptance condition α of $\mathsf{Rg}(A)$ is co-Büchi. This contradicts the fact that $\mathsf{L}(A^\pi) \subseteq \mathsf{L}_{\mathsf{coB}}(\mathsf{Rg}(A))$.

Then, any word accepted by A^π visits at most $m \times n$ times an accepting state of $\mathsf{Rg}(A)$, and $\mathsf{L}(A^\pi) \subseteq \mathsf{L}_{\mathsf{KcoB}}(\mathsf{Rg}(A))$, with $K = (2n^n n! + 1) \times |\mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)\,| \times n = (2n^{n+1}n! + n) \times |\mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)\,|$. Thus, player 1 has a winning strategy in $\langle \Sigma_1, \Sigma_2, cmax, \mathsf{L}_{\mathsf{KcoB}}(\mathsf{Rg}(A))\rangle$ too. □

Thanks to these results, we can now prove Theorem 50:

Proof of Theorem 50. Let $\overline{K} \geq (2n^{n+1}n! + n) \times |\mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)\,|$. If there is a winning strategy for player 1 in $G^T_{\overline{K}}$ then obviously there is a winning strategy for player 1 in G^T. Conversely, suppose there is a winning strategy for player 1 in G^T. Then, by Proposition 51, he has a winning strategy in $G^R = \langle \Sigma_1, \Sigma_2, cmax, \mathsf{L}_{\mathsf{coB}}(\mathsf{Rg}(A))\rangle$, and by Proposition 54, he has a winning strategy in $G^R_K = \langle \Sigma_1, \Sigma_2, cmax, \mathsf{L}_{\mathsf{KcoB}}(\mathsf{Rg}(A))\rangle$, with $K = (2n^{n+1}n! + n) \times |\mathsf{Reg}\,(\mathbb{H}_\Sigma, cmax)\,|$. By applying again Proposition 51, he has a winning strategy in the timed game $G^T_K = \langle \Sigma_1, \Sigma_2, \mathsf{L}_{\mathsf{KcoB}}(A)\rangle$. Since $K \leq \overline{K}$, $\mathsf{L}_{\mathsf{KcoB}}(A) \subseteq \mathsf{L}_{\overline{K}\mathsf{coB}}(A)$. Hence player 1 has a winning strategy in $G^T_{\overline{K}}$. □

7.6.1 Solving games defined by UECA

For solving games defined by UECA with K-co-Büchi acceptance condition, we show how to build, from a UECA $A = \langle Q, q_{in}, \Sigma, \delta, \alpha\rangle$ with K-co-Büchi acceptance condition, a DECA with 0-co-Büchi acceptance condition which is denoted $\mathsf{Det}_K(A)$, that accepts the same timed language. The construction of this DECA is based on a generalization of the *subset construction*. When applied to an untimed universal automaton A with set of locations Q, the classical subset construction consists in building a new automaton A' whose locations are subsets of Q. Thus, each location of A' encodes the set of locations of A that are active at each level of the run tree. In the case of K-co-Büchi automata, one needs to remember how many times accepting states have been visited on the branches that lead to each active location. As a consequence, the locations of the subset construction should be sets of the form $\{(q_1, n_1), \ldots, (q_\ell, n_\ell)\}$, where each q_i is an active location that has been reached by a branch visiting exactly n_i accepting states. However, in this case, the set of locations in the subset construction is not finite anymore. This can be avoided by observing that we can keep only the maximal number of visits (up to $K+1$) to accepting locations among all the branches that reach q. So, the states of the deterministic automaton are functions $F : Q \mapsto \{-1, 0, 1, \ldots, K, K+1\}$, where $F(q) = -1$ means that q is not currently active, $F(q) = k$ with $0 \leq k \leq K$ means that q is currently active and that the branch with maximal number of visits to α that leads to q has visited accepting states k times, and $F(q) = K+1$ means that q is currently active and that the branch with maximal numbers of visits to α that leads to q has visited accepting states more than K times. In this last case, the timed word

which is currently read has to be rejected, because of the K-co-Büchi condition.

Formally, $\mathsf{Det}_K(A) = \langle \mathcal{F}, F_0, \Sigma, \Delta, \alpha_K \rangle$ where the following holds. $\mathcal{F} = \{F \mid F : Q \to \{-1, 0, 1, \ldots, K, K+1\}\}$. If we let $(q \in \alpha)$ be the function that returns 1 if $q \in \alpha$ and 0 otherwise, $F_0 \in \mathcal{F}$ is such that $F_0(q_0) = (q_0 \in \alpha)$ and $F_0(q) = -1$ for all $q \in Q$ and $q \neq q_0$. Now, $\Delta(F, \sigma, \psi)$ is defined if there exists a function $h : \{q \in Q \mid F(q) \geq 0\} \to \mathsf{Constr}(\mathbb{P}_\Sigma)$ s.t. (i) ψ is equal to $\bigwedge_{q \mid F(q) \geq 0} h(q)$ and this formula is satisfiable, (ii) for all $q \in Q$ such that $F(q) \geq 0$, $\delta(q, \sigma, h(q))$ is defined. In this case, $\Delta(q, \sigma, \psi) = F'$ where F' is the counting function such that for all $q \in Q$, $F'(q)$ equals: $\max \left\{ \min\left(K+1, F(p) + (q \in \alpha)\right) \;\middle|\; q \in \delta(p, \sigma, h(p)) \wedge F(p) \neq -1 \right\}$. Finally, $\alpha_K = \{F \in \mathcal{F} \mid \exists q \in Q \cdot F(q) = K+1\}$.

Proposition 55 *For all* UECA A*, for all* $K \in \mathbb{N}$*:* $\mathsf{L}_{K\mathsf{coB}}(A) = \mathsf{L}_{0\mathsf{coB}}(\mathsf{Det}_K(A))$.

From this deterministic automaton, it is now easy to construct a *timed safety game* for solving the realizability problem. We will analyze, in details, the construction in Section 7.7 , to solve the realizability problem of a real-time extension of the logic LTL.

Remark 56 (Time divergence) *Handling time divergence in timed games requires techniques that are more involved than the ones suggested in previous sections. In the timed games considered in this section, if the set of winning plays only contains divergent timed words, then clearly player 1 can not win the game, no matter what the objective is. Indeed, as player 2 can always overtake the action proposed by player 1, he can easily block time and ensure that the output of the game is a convergent timed word. To avoid such pathological behaviors, the specification should declare player 1 winning in those cases. In [de Alfaro et al., 2003], the interested reader will find an extensive discussion on how to decide winner in the presence of time convergence.*

7.7 Safraless Algorithm for Realizability of $\mathsf{LTL}_\triangleleft$

In this section we solve the realizability problem for a fragment of the Event Clocks Logic called $\mathsf{LTL}_\triangleleft$ [Doyen et al., 2009]. For each formula of this logic, we can construct, in exponential time, a universal event-clock automaton with co-Büchi acceptance condition that accepts the set of timed words that the formula defines.

In Section 7.6 we have showed that the co-Büchi acceptance condition can be strengthened into a condition that asks that all runs of the automaton visit less than $K \in \mathbb{N}$ times the set of accepting locations, allowing to reduce the realizability problem for $\mathsf{LTL}_{\triangleleft}$ to the realizability problem for universal K-co-Büchi event-clock automata. UECA with co-Büchi acceptance condition are easily determinizable and we can reduce the original problem to a timed safety game problem that can be solved using the tool UPPAAL TIGA [Behrmann et al., 2007]. We illustrate this on a simple example.

7.7.1 Definition of $\mathsf{LTL}_{\triangleleft}$

The logic $\mathsf{LTL}_{\triangleleft}$ is a fragment of the Event Clock Logic (ECL for short) [Henzinger et al., 1998; Raskin, 1999; Raskin and Schobbens, 1999]. ECL is an extension of LTL with two real-time operators: the history operator $\triangleleft_{\mathrm{I}} \varphi$ expressing that φ was true for the last time t time units ago for some $t \in I$, and the prediction operator $\triangleright_{\mathrm{I}} \varphi$ expressing that the next time φ will be true is in t time units for some $t \in I$ (where I is an interval). $\mathsf{LTL}_{\triangleleft}$ is obtained by *disallowing prediction operators*. The realizability problem for ECL is defined as in the previous section with the exception that the set of winning plays is defined by an ECL formula instead of a UECA. The *realizability problem* has recently [Doyen et al., 2009] been shown 2EXPTIME-complete for $\mathsf{LTL}_{\triangleleft}$ but undecidable[1] for the full ECL.

We further restrict ourselves to the case where expressions of the form $\triangleleft_{\mathrm{I}} \varphi$ appear with $\varphi = a$ only, where a is some alphabet letter. Remark that this last restriction is not necessary to obtain decidability [Doyen et al., 2009], but it makes the presentation easier. Our results carry on to the more general case.

Formally, given an alphabet Σ, the syntax of $\mathsf{LTL}_{\triangleleft}$ is as follows (with $a \in \Sigma$):

$$\psi \in \mathsf{LTL}_{\triangleleft} \quad ::= \quad a \mid \neg\psi \mid \psi \vee \psi \mid \psi \,\mathcal{S}\, \psi \mid \psi \,\mathcal{U}\, \psi \mid \triangleleft_{\mathrm{I}} a$$

The models of an $\mathsf{LTL}_{\triangleleft}$ formula are infinite timed words. A timed word $\theta = (w, \tau)$ *satisfies* a formula $\varphi \in \mathsf{LTL}_{\triangleleft}$ at position $i \in \mathbb{N}$, written $\theta, i \models \varphi$, according to the following rules:

[1] Note that the undecidability proof has been made for a slightly different definition of timed games, but the proof can be adapted to the definition we rely on in the present thesis.

- if $\varphi = a$, then $w_i = a$;
- if $\varphi = \neg\varphi_1$, then $\theta, i \not\models \varphi_1$;
- if $\varphi = \varphi_1 \vee \varphi_2$, then $\theta, i \models \varphi_1$ or $\theta, i \models \varphi_2$;
- if $\varphi = \varphi_1 \,\mathcal{S}\, \varphi_2$, then there exists $0 \leq j < i$ such that $\theta, j \models \varphi_2$ and for all $j < k < i$, $\theta, k \models \varphi_1$;
- if $\varphi = \varphi_1 \,\mathcal{U}\, \varphi_2$, then there exists $j > i$ such that $\theta, j \models \varphi_2$ and for all $i < k < j$, $\theta, k \models \varphi_1$;
- if $\varphi = \lhd_I a$, then there exists $0 \leq j < i$ such that $w_j = a$, $\tau_i - \tau_j \in I$, and for all $j < k < i$, $w_k \neq a$;

When $\theta, 0 \models \varphi$, we simply write $\theta \models \varphi$ and we say that θ satisfies φ. We denote by $[\![\varphi]\!]$ the set $\{\theta \mid \theta \models \varphi\}$ of models of φ. Finally, we define the following shortcuts: $\mathsf{true} \equiv a \vee \neg a$ with $a \in \Sigma$, $\mathsf{false} \equiv \neg\mathsf{true}$, $\varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$, $\Diamond\varphi \equiv \mathsf{true}\,\mathcal{U}\,\varphi$, $\Box\,\varphi \equiv \varphi \wedge \neg\Diamond(\neg\varphi)$, $\bigcirc\,\varphi \equiv \mathsf{false}\,\mathcal{U}\,\varphi$, $\ominus\,\varphi \equiv \mathsf{false}\,\mathcal{S}\,\varphi$, and $\Diamond\!\!\!-\,\varphi \equiv \mathsf{true}\,\mathcal{S}\,\varphi$. We also freely use notations like $\geq x$ to denote the interval $[x, \infty)$, or $< x$ for $[0, x)$, etc. in the $\lhd$ operator.

Let $\Sigma = \Sigma_1 \uplus \Sigma_2$ be an alphabet that is partitioned into a set Σ_1 of player 1 events (*controllable* events), and Σ_2 of player 2 events (*uncontrollable* events), and let φ be an $\mathsf{LTL}_{\lhd}$ formula on Σ. Then, φ is *realizable* iff Player 1 has a winning strategy in the TG $\langle \Sigma_1, \Sigma_2, [\![\varphi]\!] \rangle$. The *realizability problem* for $\mathsf{LTL}_{\lhd}$ asks, given an $\mathsf{LTL}_{\lhd}$ formula φ whether φ is realizable.

7.7.2 An efficient algorithm to solve $\mathsf{LTL}_{\lhd}$ realizability

Let us now show how to exploit the results from the previous section to obtain an *incremental* algorithmic schema that solves the realizability problem of $\mathsf{LTL}_{\lhd}$. From an $\mathsf{LTL}_{\lhd}$ formula φ, we build, using standard techniques [Raskin, 1999; Raskin and Schobbens, 1999], a NECA with Büchi acceptance condition $A_{\neg\varphi}$ s.t. $\mathsf{L}_{\mathsf{B}}(A_{\neg\varphi}) = [\![\neg\varphi]\!]$. Then, we consider its dual $\tilde{A}_{\neg\varphi}$, which is thus a UECA with co-Büchi acceptance condition s.t. $\mathsf{L}_{\mathsf{coB}}(\tilde{A}_{\neg\varphi}) = [\![\varphi]\!]$. As a consequence, solving the *realizability problem* for φ now amounts to finding a winning strategy for player 1 in the timed

game $\left\langle \Sigma_1, \Sigma_2, \mathsf{L}_{\mathsf{coB}}(\tilde{A}_{\neg\varphi}) \right\rangle$. Theorem 50 tells us that we can reduce this to finding a winning strategy in a timed game whose objective is given by an automaton with K-*co-Büchi acceptance condition* (for a precise value of K). In this game, the objective of player 1 is thus to *avoid visiting accepting states too often* (no more than K times), and this is thus a *safety condition*. The automaton $\mathsf{Det}_K(\tilde{A}_{\neg\varphi})$ can be used to define a timed safety game. Such games can be solved by tools such as UPPAAL TIGA [Behrmann et al., 2007].

The drawback of this approach is that the value K is potentially intractable: it is doubly-exponential in the size of φ. As a consequence, $\mathsf{Det}_K(\tilde{A}_{\neg\varphi})$ and its underlying timed safety game are unmanageably large. To circumvent this difficulty, we adopt an incremental approach. Instead of solving the game underlying $\mathsf{Det}_K(\tilde{A}_{\neg\varphi})$, we solve iteratively the games underlying $\mathsf{Det}_i(\tilde{A}_{\neg\varphi})$ for increasing values of $i = 0, 1, \ldots$. As soon as player 1 can win a game for some i, we can stop and conclude that φ is *realizable*. Indeed, $\mathsf{L}_{0\mathsf{coB}}(\mathsf{Det}_i(\tilde{A}_{\neg\varphi})) = \mathsf{L}_{i\mathsf{coB}}(\tilde{A}_{\neg\varphi})$ by Proposition 55, and $\mathsf{L}_{i\mathsf{coB}}(\tilde{A}_{\neg\varphi}) \subseteq \mathsf{L}_{K\mathsf{coB}}(\tilde{A}_{\neg\varphi}) \subseteq [\![\varphi]\!]$. In other words, realizability of $\mathsf{L}_{0\mathsf{coB}}(\mathsf{Det}_i(\tilde{A}_{\neg\varphi}))$ implies realizability of φ. Unfortunately, if φ is *not realizable*, this approach fails to avoid considering the large theoretical bound K. To circumvent this second difficulty, we use the property that our games are determined: φ is not realizable by player 1 iff $\neg\varphi$ is realizable by player 2. So in practice, we execute two instances of our incremental algorithm in parallel and stop whenever one of the two is conclusive. The details of this incremental approach are given in [Filiot et al., 2009], and it is experimentally shown there, in the case of LTL specifications, that the values that one needs to consider for i are usually very small. To sum up, our incremental algorithm works as follows.

Fix an $\mathsf{LTL}_{\triangleleft}$ formula φ, and set i to 0. Next, **if** player 1 has a winning strategy in $\left\langle \Sigma_1, \Sigma_2, \mathsf{L}_{0\mathsf{coB}}(\mathsf{Det}_i(\tilde{A}_{\neg\varphi})) \right\rangle$, **then** φ is *realizable*; **else if** player 2 has a winning strategy in $\left\langle \Sigma_1, \Sigma_2, \mathsf{L}_{0\mathsf{coB}}(\mathsf{Det}_i(\tilde{A}_{\varphi})) \right\rangle$, **then** φ is *not realizable*; **else**, increment i by 1 and iterate.

7.7.3 Experiments with UPPAAL TIGA

We have thus reduced the realizability problem of $\mathsf{LTL}_{\triangleleft}$ to solving a sequence of TG of the form $\langle \Sigma_1, \Sigma_2, \mathsf{L}_{0\mathsf{coB}}(A) \rangle$, where A is a DECA. Solving each of these games amounts to solving a *safety game* played in an arena which is defined by A (where

the edges are partitioned according to Σ_1 and Σ_2). In practice, this can be done using UPPAAL TIGA [Behrmann et al., 2007], as we are about to show thanks to a *simple* yet *realistic* example.

Our example consists of a system where a controller monitors an input line that can be in two states: *high* or *low*. The state of the input line is controlled by the environment, thanks to the actions *up* and *down*, that respectively change the state from low to high and high to low. Changes in the state of the input line might represent *requests* that the controller has to *grant*. More precisely, whenever consecutive *up* and *down* events occur separated by at least two time units, the controller has to issue a *grant* after the corresponding *down* but before the next *up*. Moreover, successive *grants* have to be at least three time units apart, and *up* and *down* events have to be separated by at least one time unit.

This informal requirement is captured by the $\mathsf{LTL}_{\lhd}$ formula
$\varphi \equiv \mathsf{Hyp} \rightarrow \mathsf{Req}_1 \wedge \mathsf{Req}_2$ on $\Sigma = \Sigma_1 \uplus \Sigma_2$,
where $\Sigma_1 = \{grant\}$, $\Sigma_2 = \{up, down\}$ and:

$$
\begin{aligned}
\mathsf{Hyp} &\equiv \Box\Big(up \rightarrow \big(\neg down\, \mathcal{U}(down \wedge \lhd_{\geq 1} up)\big)\Big) \wedge \\
&\quad\ \Box\Big(down \rightarrow \big(\neg up\, \mathcal{U}(up \wedge \lhd_{\geq 1} down)\big)\Big) \\
\mathsf{Req}_1 &\equiv \Box\big((down \wedge \lhd_{>2} up) \rightarrow (\neg up\, \mathcal{U}\, grant)\big) \\
\mathsf{Req}_2 &\equiv \Box(grant \rightarrow \neg \lhd_{<3} grant)
\end{aligned}
$$

Remark that φ does not forbid the controller from producing *grant* events that have not been requested by the environment. However, a controller producing *grants too often* might hinder itself because Req_2 requires each pair of grants to be separated from each other by at least 3 time units.

We illustrate this by showing two prefixes of executions in Fig. 7.1 and Fig. 7.2. The state of the input is represented on top, grants are represented at the bottom. Each dot represents a *grant* event. Thick lines represent the period during which the controller cannot produce any *grant* because of Req_2. In Fig. 7.1 we show a prefix that respects φ. In Fig. 7.2 we show a case where the controller has issued an unnecessary grant that prevents him from granting the request that appears with the *down* event at time 5.75.

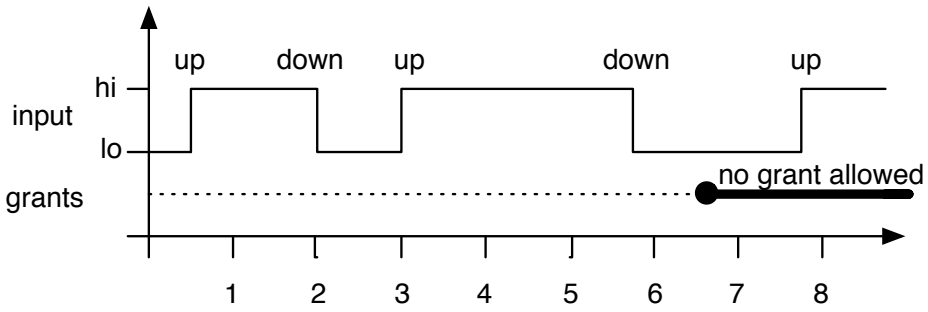

Figure 7.1: *An example of execution of the system that respects* φ

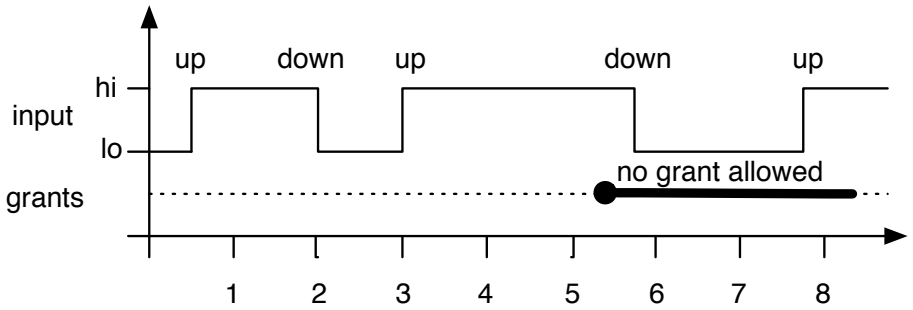

Figure 7.2: *An example of execution of the system that does not respects* φ

Let us now apply the algorithmic schema presented above to this example. We first build the NECA with Büchi acceptance condition $A_{\neg\varphi}$, given in Fig. 7.3. This automaton has two parts, identified by the names of the states: the top part (corresponding to the states $1, \ldots 7$) accepts the models of $[\![\neg(\mathsf{Hyp} \rightarrow \mathsf{Req}_1)]\!]$ and the lower part (states $1, \overline{2}, \ldots, \overline{6}$) accepts the models of $[\![\neg(\mathsf{Hyp} \rightarrow \mathsf{Req}_2)]\!]$, so the whole automaton accepts exactly $[\![\neg\varphi]\!]$. Fig. 7.3 can also be regarded as a depiction of the dual UECA with co-Büchi acceptance condition $\tilde{A}_{\neg\varphi}$, by interpreting non-determinism as universal branching.

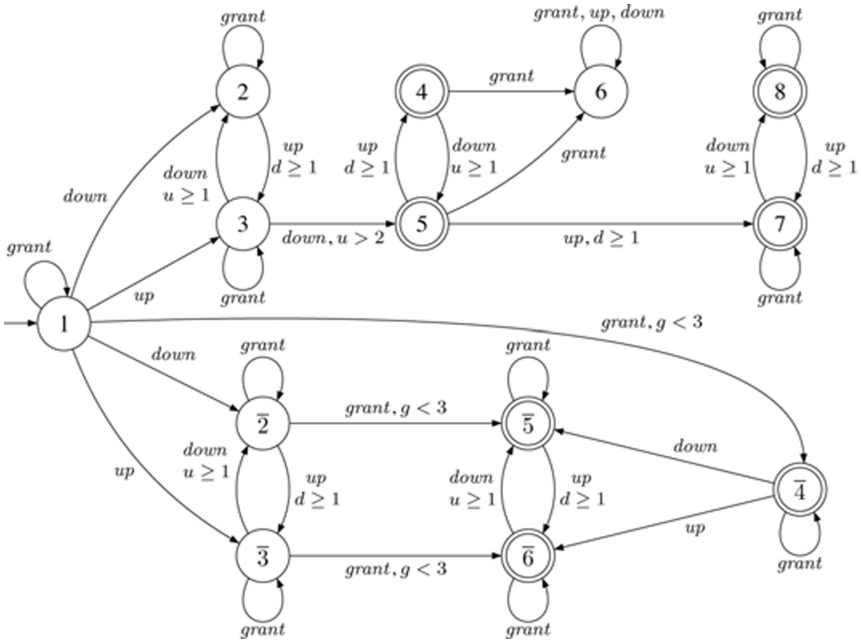

Figure 7.3: *The* NECA $A_{\neg\varphi}$

From $\tilde{A}_{\neg\varphi}$, we have applied the counting functions construction described above, for $i = 1$. In order to ease the presentation, we have applied this construction separately on the two parts of the automaton, to obtain G_1 and G_2, the DECA given in Fig. 7.4. These automata are shown as they appear in their UPPAAL TIGA encoding: controllable transitions are plain, and uncontrollable transitions are dashed. The history clocks corresponding to *up*, *down* and *grant* are respectively denoted u, d and g. Unreachable states, as well as transitions to the state F with $F(q) = -1$ for any q are not shown. Remark that since UPPAAL TIGA uses classical Alur-Dill timed automata, and not ECA, we have to explicitly manage the reset of those clocks. Finally, observe that we have used the synchronisation mechanism offered by UPPAAL TIGA to ensure that the game is played on the synchronous product of these two automata (which corresponds to the counting function construction applied to $A_{\neg\varphi}$).

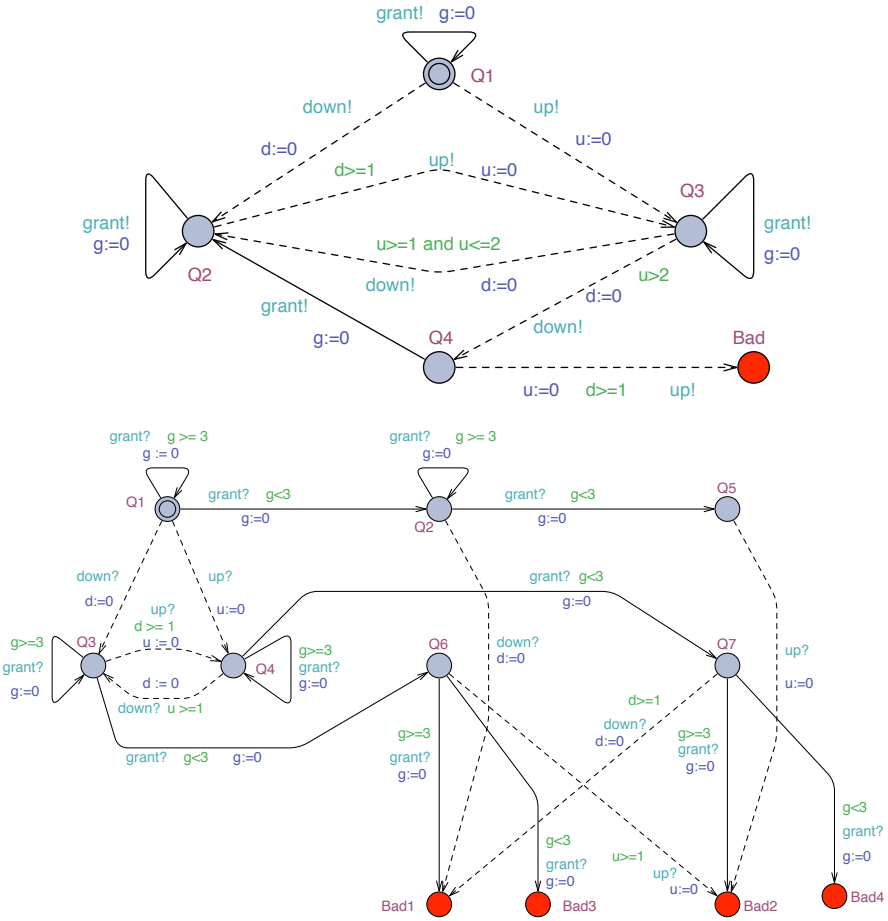

Figure 7.4: *The* DECA *obtained from the two parts of* $A_{\neg\varphi}$

We provided this model to UPPAAL TIGA together with the synthesis objective `control: A[not BadState]`, where `BadState` is true iff one of the automata reaches one of its Bad locations (that corresponds to one of the counters being > 1). In this case, UPPAAL TIGA can compute a *winning strategy* for player 1, which means that player 1 is capable of ensuring that, on any branch of any run of $\tilde{A}_{\neg\varphi}$, accepting states occur at most one time. This strategy thus ensures that all the plays are accepted by $\tilde{A}_{\neg\varphi}$, and so they all satisfy φ. Hence, φ is realizable.

This example shows that, although an exponentially-large K might be needed to prove realizability of an $\mathsf{LTL}_{\triangleleft}$ formula, in practice, small values of i (here, 1) might be sufficient.

A larger set of experiments (on large LTL formulas) exploiting the same techniques can be found in [Filiot et al., 2009]. These experiments confirm that small values of i are sufficient in practice.

Remark 57 (Time divergence) *In this example, time divergence is not an issue. Indeed, the objective is such that, on the one hand, player 1 wins the game if player 2 proposes to play up followed by down, or down followed by up without waiting at least one time unit (because of* Hyp*), and, on the other hand, player 1 violates* Req_2 *if he plays two grant actions too close in time (less than 3 t.u. apart).*

Chapter 8

Conclusions

The Verification and Synthesis problems for parametric and real-time systems have been considered as one of the most ambitious and challenging problems in system design.

In the first part of this thesis we have examined the Verification problem, investigating several aspects of introducing parametric constants to express timing constraints in real-time linear-time temporal logics. The possibility of using such parametric constants is of great appeal mostly in the early stages of a design when, due to the scarce information on the system, the exact value of these constants is hard, or even impossible, to determine. In particular, we have proposed the logic PMITL as a parametric extension of MITL. This logic turned out to be decidable and within the same computational complexity class as MITL.

In fact, we have shown that the satisfiability, validity and model-checking problems are all EXPSPACE-complete within this formalism, using a translation to the emptiness and the universality problems for Büchi L/U automata.

To get the whole picture of the PMITL analysis, we have focused on the study of the computational complexity of natural syntactic fragments of PMITL, showing that in meaningful fragments of the logic the considered decision problems are PSPACE-complete.

In these regards, the impossibility of characterizing the space of the fulfilling parameter valuations has motivated the formulation of a general decision problem which gives more information on this space than simply considering the emptiness and universality problems. We have considered a remarkable problem expressed by queries

where the values that each parameter may assume are either existentially or universally quantified. We have solved this problem in several cases and exhibited an algorithm in EXPSPACE.

In the second part of this thesis we have considered the Synthesis problem for real-time systems, investigating the applicability of automata constructions that avoid determinization for solving the language inclusion problem and the realizability problem for real-time logics.

Since Safra's determinization procedure is difficult to implement, we have presented extensions of Safraless algorithms proposed in the literature [Filiot et al., 2009; Kupferman and Vardi, 2001, 2005b; Schewe and Finkbeiner, 2007] for automata on infinite untimed words to the case of automata on infinite timed words. More precisely, we have introduced Safraless procedures for the two following problems:

1. computing the complement of an alternating event-clock automaton with Büchi acceptance condition.

2. solving timed games whose objectives are specified by Past alternating event clock automata with co-Büchi acceptance condition.

Regarding the first problem, we have shown that the techniques of Kupferman and Vardi [2001] can be adapted to alternating event-clock automata. The procedure introduced have been used to complement nondeterministic event-clock automata with Büchi acceptance conditions, leading to algorithms for solving the universality and language inclusion problems for that class of timed automata without resorting to the Safra's construction.

Regarding the second problem, we have generalized the ideas of Filiot et al. [2009] to solve the realizability problem for a fragment of the Event Clocks Logic called $\mathsf{LTL}_{\triangleleft}$ [Doyen et al., 2009]. We have shown how our techniques can be used to reduce the original problem to solving a sequence of timed safety games, which can be done in practice thanks to tools such as UPPAAL TIGA TiGa [Behrmann et al., 2007].

8.1 Future works

It is interesting to go into more depth for the decision problems we have introduced in this thesis. Starting from our results, there are several open problems that could be analyzed. We can resume the future works towards the following directions.

Characterization of parameter valuations domain In this thesis we have generalized the problem of characterizing the space of the fulfilling parameter valuations by considering general queries on such spaces, where each parameter is either quantified existentially or universally. We have solved this query decision problem with the restriction that either all the lower bound parameters are existentially quantified or all the upper bound parameters are existentially quantified. Note that even if this does not cover all the cases, it gives us a feedback on the fulfilling valuations of formulas with both kinds of parameters, and thus covers cases where the characterization of this space does not look possible.

We leave open the problem when general queries are allowed. It is also open the generalization of these results when parametric linear expressions are allowed as right endpoints of the intervals. In this case, the main problem seems to be the fact that the admissible values for a parameter are depending on the values assigned to the other parameters, and therefore, it looks hard to show that the inversion of quantifiers does not alter the meaning of a query.

Besides the above open problems, another interesting future research direction is to explicitly constrain the space of the admissible valuations with Boolean combinations of linear systems over the parameters. Such a study has been already addressed for Büchi L/U automata by Bozzelli and La Torre [2009]. Concerning to the universality and emptiness problems, we are able to show the same results as in [Bozzelli and La Torre, 2009] within our formalism. However, it is not clear if the extension of these results to the query decision problem is feasible.

Expressiveness of the fragments of PMITL The expressiveness of the fragments of PMITL we have considered requires further investigation. In particular, our results show that the fragments $\mathrm{P}_{0,\infty}$MITL and $\mathrm{PMITL}_{0,\infty}$ are both equivalent to PMITL. However we do not know if $\mathrm{P}_{0,\infty}\mathrm{MITL}_{0,\infty}$ can express to the whole PMITL. For

the non-parameterized analogous of these logic, i.e., $\text{MITL}_{0,\infty}$ and MITL, equivalence holds though MITL is exponentially more succinct than $\text{MITL}_{0,\infty}$ [Alur et al., 1996]. The translation given there uses negation, and therefore, it does not seem to be suitable for obtaining a similar translation in the parameterized settings.

Parametric extensions of branching-time timed logics In this thesis we have considered linear-time logics, where a system is viewed as a set of runs and formulas express properties over these runs. To express specification properties it is also used the branching time framework, where a system is viewed as a tree and each branch of this tree represents a computation of the system. In branching-time timed logics formulas are interpreted over states having several possible successors and we can quantify existentially or universally over the different possible futures of a given state. These formalisms differ from an expressiveness point of view, and the model checking algorithms are also very different.

The most popular (untimed) branching-time temporal logic is CTL(computation tree logic) [Clarke and Emerson, 1981]. Classical CTL is a modal logic used to reason about the temporal behavior of systems considering either *all the possible futures* or *at least one possible future*. It contains the modalities "always" ($\forall\Box$), "potentially" ($\exists\Diamond$), "exists-until" ($\exists\,\mathcal{U}$), and "for-all-until" ($\forall\,\mathcal{U}$). For example, the formula $\exists\varphi_1\,\mathcal{U}\,\varphi_2$ holds in a state q if and only if there exists a path from q where φ_2 is true at some position q', and φ_1 is true at any position between q and q'. To have a complete picture about temporal logics for the specification and verification of reactive systems, see [EME 91].

In the context of real-time systems, timed extensions of CTL, as TCTL, have been studied by many authors [Alur et al., 1993a; La Torre and Napoli, 2003]. Moreover, parametric branching-time specifications were first investigated in [Emerson and Trefler, 1999; Wang, 1996] where decidability is shown for logics obtained as extensions of TCTL [Alur et al., 1993a] with parameters.

Recently, several researchers as Kupferman et al. [2002] and Ferrante et al. [2009a] (preliminary version in [Ferrante et al., 2008]) have considered generalizations of branching-time logics with the addition of *graded* modalities that generalize standard quantifiers. While classical branching time logics can be used for reasoning about the temporal behavior of systems, considering either all the possible futures or at least one

possible future, graded logics use graded modalities that allow to reasoning about more than a given number of possible distinct future behaviors. Thus, this new logics allow to express interesting properties, such as the safety property that "a system always has at least two ways to reach a *safe state*" ($\forall \Box \exists^{>1} \Diamond$ safe). Clearly, formulas of this type cannot be expressed in CTL and not even in classical μ-calculus.

An interesting future research direction is then to study the effect, both in the expressiveness and in the complexity of the decision problems, of the introduction of parameters in the generalizations of branching-time logics with graded modalities.

Observe that the motivation in the use of these graded modalities is practical. It mainly arises from the fact that, during the verification of a system design, a central feature of the model-checking technique is the generation of counterexamples, and a model checking tool against a graded formula can generate more counterexamples in each run. For graded-CTL, that is the logic obtained augmenting CTL with graded modalities, a tool with good performances has then been developed, extending the known tool Nu-SMV [Ferrante et al., 2009b].

Another possible future direction to work on, is to verify, in practice, whether a parametric model-checker tool could be augmented with these graded modalities, retaining the usual performances.

Parametric games Two-player graph games of infinite duration are a tool to synthesize controllers for reactive systems, i.e., systems which have to interact with an (possibly antagonistic) environment. Requirements on the controlled system are typically given by a subset of the systems executions. The controller has to react to the moves of the environment in a way such that the execution satisfies the requirement. The requirements are usually given by acceptance conditions from the theory of automata on infinite words. However, in practice, it is often more convenient to work in a logic framework.

A concise way to specify requirements on infinite executions is to use linear temporal logic (LTL). The problem is that LTL lacks capabilities to express timing constraints, which are often desirable in applications. In order to express quantitative information, there were introduced extensions of LTL with timing constraints and extensions of LTL with variable bounds for model checking. An example is given by the logic PLTL, introduced by Alur et al. [2001] and by PROMPT-LTL, introduced by Kupferman et al.

[2009].

In a recent work on Parametric LTL games, Zimmermann [2010] lifts the results on PLTL to graph-based games: he presents algorithms to determine whether a player wins a PLTL game with respect to some, infinitely many, or all variable valuations. For winning conditions with only parameterized eventualities $\Diamond_{\leq x}$ or only parameterized always $\Box_{\leq y}$ operators, solving games can be seen as an optimization problem. We wonder which is the best variable valuation such that a player wins a given game with respect to that valuation.

The alternating-color technique presented by Kupferman et al. [2009] allows a comprehensive treatment of PROMPT-LTL: it is used to solve the model checking and assume-guarantee model checking problem, as well as the realizability problem, an abstract game given only by a winning condition ϕ, but without an underlying game graph.

Zimmermann [2010] first applies the alternating-color technique to graph-based PROMPT-LTL games, thereby transferring the results on realizability of PROMPT-LTL specifications to this domain. Then, he is able to solve the problems for PLTL, employing the results on PROMPT-LTL games at several points. As model checking can be seen as a one-player game, his results also give a simpler proof of the results on PLTL model checking.

All the algorithms presented by Zimermann run in doubly-exponential time, which is asymptotically optimal, as solving classical LTL games is 2EXPTIME-complete. Hence, adding bounded temporal operators to LTL does not increase the complexity of solving games with winning conditions in the extended logics. This confirms similar findings on PLTL model checking.

The first interesting direction to work on is the introduction of other kinds of intervals that are not considered by Zimmermann [2010], where the only parametric interval is of the form $(0, k)$, for a parameter k. In particular, we are investigating if it is possible to apply the alternating-color technique even if we introduce the intervals presented in PMITL. We should focus on the analysis of the length of the blocks where we consider the validity of the subformulas used in the construction. Firstly, we try to apply the techniques of Zimermann for non-singular time intervals with end-points in $\mathbb{N} \cup \{\infty\}$ and after that we take in consideration parameterized time intervals, defined in Chapter 4.

Another interesting direction to work on is the possibility of applying the techniques, showed by Zimermann for PLTL games, to solve $\mathsf{PLTL}_{\triangleleft}$ games.

The logic $\mathsf{PLTL}_{\triangleleft}$ we consider is obtained from the introduction of parameters in the logic $\mathsf{LTL}_{\triangleleft}$ studied in Chapter 7. We introduce a parametric real-time operators: the history operator $\triangleleft_{\mathrm{I_k}} \varphi$ expressing that φ was true for the last time t time units ago for some $t \in I_k$.

The parameter k is considered in the semantics of the classical real-time operator $\triangleleft_{\mathrm{I_k}} \varphi$, to express a lower bound for the last time φ was true.

Formally, given an alphabet Σ, the syntax of $\mathsf{PLTL}_{\triangleleft}$ is as follows (with $a \in \Sigma$):

$$\psi \in \mathsf{PLTL}_{\triangleleft} \quad ::= \quad a \mid \neg\psi \mid \psi \vee \psi \mid \psi \,\mathcal{S}\, \psi \mid \psi \,\mathcal{U}\, \psi \mid \triangleleft_{\mathrm{I}} a \mid \triangleleft_{\mathrm{I_k}} a$$

The models of an $\mathsf{PLTL}_{\triangleleft}$ formula are infinite timed words.

The semantics of the $\mathsf{PLTL}_{\triangleleft}$ logic is defined in the same way we have already seen for $\mathsf{LTL}_{\triangleleft}$ logic in Chapter 7.

We denote by $[\![\varphi]\!]$ the set $\{\theta \mid \theta \models \varphi\}$ of models of φ.

Let $\Sigma = \Sigma_1 \uplus \Sigma_2$ be an alphabet that is partitioned into a set Σ_1 of player 1 events (*controllable* events), and Σ_2 of player 2 events (*uncontrollable* events), and let φ be an $\mathsf{LTL}_{\triangleleft}$ formula on Σ. Then, φ is *realizable* iff Player 1 has a winning strategy in the TG $\langle \Sigma_1, \Sigma_2, [\![\varphi]\!] \rangle$. The *realizability problem* for $\mathsf{PLTL}_{\triangleleft}$ asks, given an $\mathsf{PLTL}_{\triangleleft}$ formula φ whether there exists a parameter valuation v such that φ is realizable.

We remark that $\mathsf{PLTL}_{\triangleleft}$, with respect to a fixed valuation v, is no more expressive than $\mathsf{LTL}_{\triangleleft}$ (albeit more succinct).

In conclusion, we would like to solve $\mathsf{PLTL}_{\triangleleft}$ games, using an extensions of the techniques presented by Zimmermann [2010].

References

Martín Abadi and Leslie Lamport. An old-fashioned recipe for real time. In *REX Workshop*, pages 1–27, 1991. 40

Martín Abadi, Leslie Lamport, and Pierre Wolper. Realizable and unrealizable specifications of reactive systems. In *ICALP*, pages 1–17, 1989. 32

S. Aggarwal and Robert P. Kurshan. Automated implementation from formal specification. In *PSTV*, pages 127–136, 1984. 94

Christoph Schulte Althoff, Wolfgang Thomas, and Nico Wallmeier. Observations on determinization of Büchi automata. In *CIAA*, pages 262–272, 2005. 126

Rajeev Alur. *Techniques for automatic verification of Real-time Systems*. PhD thesis, Stanford University, Stanford, 1991. 2, 17, 24, 40

Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In *ICALP*, pages 322–335, 1990. 41

Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126 (2):183–235, 1994. 21, 22, 36, 60, 107, 108

Rajeev Alur and Thomas A. Henzinger. Logics and models of real time: a survey. In *REX Workshop*, pages 74–106, 1991. 25, 33, 34

Rajeev Alur and Thomas A. Henzinger. Back to the future: towards a theory of timed regular languages. In *FOCS*, pages 177–186, 1992. 41

Rajeev Alur and Thomas A. Henzinger. Real-time logics: complexity and expressiveness. *Inf. Comput.*, 104(1):35–77, 1993. 33, 34, 35, 36, 41, 44

Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *J. ACM*, 41(1):181–204, 1994. 33, 35, 41

Rajeev Alur and Robert P. Kurshan. Timing analysis in Cospan. In *Hybrid Systems*, pages 220–231, 1995. 22

Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking for real-time systems. In *LICS*, pages 414–425, 1990. 41

Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. *Inf. Comput.*, 104(1):2–34, 1993a. 22, 151

Rajeev Alur, Costas Courcoubetis, and Thomas A. Henzinger. Computing accumulated delays in real-time systems. In *CAV*, pages 181–193, 1993b. 22

Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *STOC*, pages 592–601, 1993c. 8, 39, 40, 42, 51

Rajeev Alur, Limor Fix, and Thomas A. Henzinger. A determinizable class of timed automata. In *CAV*, pages 1–13, 1994. 22

Rajeev Alur, Alon Itai, Robert P. Kurshan, and Mihalis Yannakakis. Timing verification by successive approximation. *Inf. Comput.*, 118(1):142–157, 1995. 22

Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, 1996. 7, 22, 33, 34, 35, 41, 44, 53, 54, 64, 67, 69, 70, 71, 75, 76, 77, 78, 151

Rajeev Alur, Kousha Etessami, Salvatore La Torre, and Doron Peled. Parametric temporal logic for "model measuring". *ACM Trans. Comput. Log.*, 2(3):388–407, 2001. 8, 39, 40, 43, 45, 63, 64, 75, 88, 89, 91, 152

Hagit Attiya, Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. In *STOC*, pages 359–369, 1991. 41

Ilan Beer, Shoham Ben-David, Daniel Geist, Raanan Gewirtzman, and Michael Yoeli. Methodology and system for practical formal verification of reactive hardware. In *CAV*, pages 182–193, 1994. 47

Gerd Behrmann, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. UPPAAL-Tiga: time for playing games! In *CAV*, pages 121–125, 2007. 12, 131, 140, 142, 143, 149

Johan Bengtsson, Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL - a tool suite for automatic verification of real-time systems. In *Hybrid Systems*, pages 232–243, 1995. 22

Laura Bozzelli and Salvatore La Torre. Decision problems for lower/upper bound parametric timed automata. *Formal Methods in System Design*, 35(2):121–151, 2009. 40, 43, 44, 49, 52, 63, 64, 70, 74, 75, 76, 77, 85, 88, 150

Véronique Bruyère and Jean-François Raskin. Real-time model-checking: parameters everywhere. In *FSTTCS*, pages 100–111, 2003. 8, 39

J. Richard Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, page 6692, 1960. 125

J. Richard Büchi. On a decision method in restricted second-order arithmetics. In *In International Congress on Logic, Method and Philosophy of Science*, page 112, 1962. 20, 21, 95, 96, 99, 125

J.Richard Büchi. *Decidable Theories II- The Monadic second-order theory of* ω_1. Lecture Notes in Mathematics, Springer-Verlag, 1973. 96

Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28 (1):114–133, 1981. 98

Remy Chevallier, Emmanuelle Encrenaz-Tiphène, Laurent Fribourg, and Weiwen Xu. Timed verification of the generic architecture of a memory circuit using parametric timed automata. *Formal Methods in System Design*, 34(1):59–81, 2009. 58, 59, 60

Yaacov Choueka. Theories of automata on ω-tapes: a simplified approach. *J. Comput. Syst. Sci.*, 8(2):117–141, 1974. 96

A. Church. Logic, arithmetic and automata. In *International Congress of Mathematicians*, pages 23–35, 1962. 6, 32, 117

Robert Clarisó and Jordi Cortadella. The Octahedron abstract domain. In *SAS*, pages 312–327, 2004a. 59

Robert Clarisó and Jordi Cortadella. Verification of timed circuits with symbolic delays. In *ASP-DAC*, pages 628–633, 2004b. 59

Edmund M. Clarke. The birth of model checking. In *25 Years of Model Checking*, pages 1–26, 2008. 4

Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs*, pages 52–71, 1981. 122, 151

Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986. 92

Edmund M. Clarke, I. A. Draghicescu, and Robert P. Kurshan. A unified approch for showing language inclusion and equivalence between various types of ω -automata. *Inf. Process. Lett.*, 46(6):301–308, 1993. 100

Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 2000. 4

Costas Courcoubetis and Mihalis Yannakakis. Minimum and maximum delay problems in real-time systems. *Formal Methods in System Design*, 1(4):385–415, 1992. 8, 22, 39

Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252, 1977. 59

Conrado Daws, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. The tool KRONOS. In *Hybrid Systems*, pages 208–219, 1995. 22

Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. The element of surprise in timed games. In *CONCUR*, pages 142–156, 2003. 139

Giuseppe De Giacomo and Maurizio Lenzerini. Concept language with number restrictions and fixpoints, and its relationship with μ-calculus. In *ECAI*, pages 411–415, 1994. 125

Giovanni De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994. 6

Barbara Di Giampaolo, Gilles Geeraerts, Jean-François Raskin, and Nathalie Sznajder. Safraless procedures for timed specifications. In *FORMATS*, volume 6246 of *Lecture Notes in Computer Science*, pages 2–22. Springer, 2010a. 11, 101, 131

Barbara Di Giampaolo, Salvatore La Torre, and Margherita Napoli. Parametric metric interval temporal logic. In *LATA*, volume 6031 of *Lecture Notes in Computer Science*, pages 249–260. Springer, 2010b. 11, 40, 53

David L. Dill. *Trace theory for automatic hierarchical verification of speed independent circuits*. MIT Press, 1989a. 32

David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems*, pages 197–212, 1989b. 41

Laurent Doyen and Jean-François Raskin. Antichain algorithms for finite automata. In *TACAS*, pages 2–22, 2010. 113

Laurent Doyen, Gilles Geeraerts, Jean-François Raskin, and Julien Reichert. Realizability of real-time logics. In *FORMATS*, pages 133–148, 2009. 7, 9, 11, 118, 131, 139, 140, 149

Doron Drusinsky and David Harel. On the power of bounded concurrency I: finite automata. *J. ACM*, 41(3):517–539, 1994. 98

Jacob Elgaard, Nils Klarlund, and Anders Møller. Mona 1.x: New techniques for WS1S and WS2S. In *CAV*, pages 516–520, 1998. 126

Calvin C. Elgot. *Decision problems of finite-automata design and related arithmetics*. Trans. Amer. Math. Soc., 1961. 125

E. Allen Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science, Volume B: formal Models and Sematics (B)*, pages 995–1072. 1990. 17, 44

E. Allen Emerson and Edmund M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program.*, 2(3):241–266, 1982. 117

E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs (extended abstract). In *FOCS*, pages 328–337, 1988. 125

E. Allen Emerson and Charanjit S. Jutla. Tree automata, μ-calculus and determinacy. In *FOCS*, pages 368–377, 1991. 95, 110, 125

E. Allen Emerson and A. Prasad Sistla. Deciding full branching time logic. *Information and Control*, 61(3):175–201, 1984. 96

E. Allen Emerson and Richard J. Trefler. Parametric quantitative temporal reasoning. In *LICS*, pages 336–343, 1999. 8, 39, 151

E. Allen Emerson, Aloysius K. Mok, A. Prasad Sistla, and Jai Srinivasan. Quantitative temporal reasoning. In *CAV*, pages 136–145, 1990. 36, 41, 47

Alessandro Ferrante, Margherita Napoli, and Mimmo Parente. CTL model-checking with graded quantifiers. In *ATVA*, pages 18–32, 2008. 151

Alessandro Ferrante, Margherita Napoli, and Mimmo Parente. Model checking for Graded CTL. *Fundam. Inform.*, 96(3):323–339, 2009a. 151

Alessandro Ferrante, Margherita Napoli, and Mimmo Parente. Graded-CTL: satisfiability and symbolic model checking. In *ICFEM*, pages 306–325, 2009b. 152

Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. An antichain algorithm for LTL realizability. In *CAV*, pages 263–277, 2009. 9, 124, 128, 129, 130, 131, 142, 147, 149

Michael J. Fischer and Lenore D. Zuck. Reasoning about uncertainty in fault-tolerant distributed systems. In *FTRTFT*, pages 142–158, 1988. 116

R.W. Floyd. Assigning meaning to programs. In *Proceedings Symposium on Applied Mathematics*, page 1932, 1967. 116

Ehud Friedgut, Orna Kupferman, and Moshe Y. Vardi. Büchi complementation made tighter. *Int. J. Found. Comput. Sci.*, 17(4):851–868, 2006. 95

Joyce Friedman. Some results in Church's restricted recursive arithmetic. *J. Symb. Log.*, 22(4):337–342, 1957. 6

Dov M. Gabbay. Applications of trees to intermediate logics. *J. Symb. Log.*, 37(1): 135–138, 1972. 125

Dov M. Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal basis of fairness. In *POPL*, pages 163–173, 1980. 24

Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: a Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*, 2002. Springer. ISBN 3-540-00388-6. 130

Sankar Gurumurthy, Roderick Bloem, and Fabio Somenzi. Fair simulation minimization. In *CAV*, pages 610–624, 2002. 94

Sankar Gurumurthy, Orna Kupferman, Fabio Somenzi, and Moshe Y. Vardi. On complementing nondeterministic Büchi automata. In *CHARME*, pages 96–110, 2003. 95

R.H. Hardin, Z. Harel, and Robert P. Kurshan. COSPAN. In *In Proc. 8th CAV, LNCS*, page 423427, 1996. 94

David Harel and Orna Kupferman. On object systems and behavioral inheritance. *IEEE Trans. Software Eng.*, 28(9):889–903, 2002. 94

David Harel and Amir Pnueli. *On the development of reactive systems*. Springer-Verlag, Berlin/New York, 1985. 116

Thomas A. Henzinger. *The temporal specification and verification of real-time systems*. PhD thesis, Stanford University, Stanford, 1991. 33, 35, 40

Thomas A. Henzinger. It's about time: real-time logics reviewed. In *CONCUR*, pages 439–454, 1998. 33, 34

Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. What good are digital clocks? In *ICALP*, pages 545–558, 1992. 34

Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. Temporal proof methodologies for timed transition systems. *Inf. Comput.*, 112(2):273–337, 1994a. 23

Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Inf. Comput.*, 111(2):193–244, 1994b. 22, 41

Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. A user guide to HyTech. In *TACAS*, pages 41–71, 1995. 60

Thomas A. Henzinger, Jean-François Raskin, and Pierre-Yves Schobbens. The regular real-time languages. In *ICALP*, pages 580–591, 1998. 33, 140

Yoram Hirshfeld and Alexander Moshe Rabinovich. Logics for real time: decidability and complexity. *Fundam. Inform.*, 62(1):1–28, 2004. 34

C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12 (10):576–580, 1969. 116

C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985. ISBN 0-13-153271-5. 116

Gerard J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall International Editions, 1991. 100

Gerard J. Holzmann. The model checker SPIN. *IEEE Trans. Software Eng.*, 23(5): 279–295, 1997. 95

Gerard J. Holzmann and Margaret H. Smith. Software model checking. In *FORTE*, pages 481–497, 1999. 4

Jozef Hooman. *Specification and Compositional Verification of Real-Time Systems*, volume 558 of *Lecture Notes in Computer Science*. Springer, 1991. ISBN 3-540-54947-1. 40

John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979. ISBN 0-201-02988-X. 20

Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits W. Vaandrager. Linear parametric model checking of timed automata. *J. Log. Algebr. Program.*, 52-53:183–220, 2002. 8, 39, 40, 42, 52, 64, 88

Farnam Jahanian. Verifying properties of systems with variable timing constraints. In *IEEE Real-Time Systems Symposium*, pages 319–329, 1989. 41

Barbara Jobstmann. *Applications and Optimizations for* LTL *Synthesis*. PhD thesis, University of Technology, Austria, 2007. 6, 117

Barbara Jobstmann and Roderick Bloem. Optimizations for LTL synthesis. In *FMCAD*, pages 117–124, 2006. 129

Marcin Jurdzinski. Small progress measures for solving parity games. In *STACS*, pages 290–301, 2000. 126

Yonit Kesten and Amir Pnueli. Verification by augmented finitary abstraction. *Inf. Comput.*, 163(1):203–243, 2000. 94

Yonit Kesten, Nir Piterman, and Amir Pnueli. Bridging the gap between fair simulation and trace inclusion. *Inf. Comput.*, 200(1):35–61, 2005. 94

Nils Klarlund. *Progress Measures and Finite Arguments for Infinite Computations*. PhD thesis, Cornell University, New York, 1990. 97, 101

Nils Klarlund. Progress measures for complementation of ω -automata with applications to temporal logic. In *FOCS*, pages 358–367, 1991. 95, 97, 100, 101

Nils Klarlund and Dexter Kozen. Rabin measures and their applications to fairness and automata theory. In *LICS*, pages 256–265, 1991. 97

Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990. 33, 36, 38, 40, 44

Dexter Kozen. Results on the propositional μ-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983. 125

Dexter Kozen and Rohit Parikh. A decision procedure for the propositional μ-calculus. In *Logic of Programs*, pages 313–325, 1983. 125

Saul A. Kripke. Semantical considerations on modal logic. *Acta Philosophica Fennica*, 16:8394, 1963. 16

Fred Kröger. *Temporal Logic of Programs*. Springer-Verlag, Berlin, 1987. 17

Orna Kupferman and Moshe Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Log.*, 2(3):408–429, 2001. 9, 11, 93, 95, 97, 99, 100, 101, 110, 111, 149

Orna Kupferman and Moshe Y. Vardi. From complementation to certification. In *TACAS*, pages 591–606, 2004. 95

Orna Kupferman and Moshe Y. Vardi. Complementation constructions for nondeterministic automata on infinite words. In *TACAS*, pages 206–221, 2005a. 94

Orna Kupferman and Moshe Y. Vardi. Safraless decision procedures. In *FOCS*, pages 531–542, 2005b. 9, 118, 124, 126, 127, 129, 130, 149

Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *J. ACM*, 47(2):312–360, 2000. 99, 117

Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. Module checking. *Inf. Comput.*, 164(2):322–344, 2001. 115

Orna Kupferman, Ulrike Sattler, and Moshe Y. Vardi. The complexity of the Graded μ–calculus. In *CADE-18: Proceedings of the 18th International Conference on Automated Deduction*, pages 423–437, London, UK, 2002. Springer-Verlag. ISBN 3-540-43931-5. 151

Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. Safraless compositional synthesis. In *CAV*, pages 31–44, 2006. 118

Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. From liveness to promptness. *Formal Methods in System Design*, 34(2):83–103, 2009. 44, 47, 152, 153

Robert P. Kurshan. *Computer Aided Verification of Coordinating Processes: the automata-theoretic approach*. Princeton University Press, 1994. 93, 94, 100

Salvatore La Torre and Margherita Napoli. Finite automata on timed omega-trees. *Theor. Comput. Sci.*, 293(3):479–505, 2003. 151

Leslie Lamport. "sometime" is sometimes "not never" - on the temporal logic of programs. In *POPL*, pages 174–185, 1980. 116

Leslie Lamport. A fast mutual exclusion algorithm. *ACM Trans. Comput. Syst.*, 5(1): 1–11, 1987. 42

Harry R. Lewis. A logic of concrete time intervals (extended abstract). In *LICS*, pages 380–389, 1990. 41

Orna Lichtenstein and Amir Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *POPL*, pages 97–107, 1985. 31, 92

Peter A. Lindsay. On alternating ω -automata. *J. Comput. Syst. Sci.*, 36(1):16–24, 1988. 98, 99

Ming T. Liu. Protocol engineering. *Advances in Computers*, 29:79–195, 1989. 92

Christof Löding and Wolfgang Thomas. Alternating automata and logics over infinite words. In *IFIP TCS*, pages 521–535, 2000. 101

Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems: Specification*. Springer, 1992a. 46, 92, 117

Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems-Specification*. Springer-Verlag, New York, 1992b. 17

Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems-Safety properties*. Springer-Verlag, New York, 1995. 17

Zohar Manna and Pierre Wolper. Synthesis of communicating processes from temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 6(1):68–93, 1984. 117, 122

René Mazala. Infinite games. In *Automata, Logics, and Infinite Games*, pages 23–42, 2001. 119

Robert McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9(5):521–530, 1966. 96

Stephan Merz. Model checking: a tutorial overview. In *MOVEP*, pages 3–38, 2000. 1, 4, 15

Albert R. Meyer. Weak monadic second order theory of successor is not elementary recursive. In *Lecture Notes in Mathematics*, page 132154, 1975. 125

M. Michel. Complementation is more difficult with automata on infinite words. In *CNET*, 1988. 95, 97, 98, 99, 101

Satoru Miyano and Takeshi Hayashi. Alternating finite automata on ω -words. In *CAAP*, pages 195–210, 1984. 98, 100, 102, 112

Faron Moller and Graham M. Birtwistle, editors. *Logics for Concurrency - Structure versus Automata (8th Banff Higher Order Workshop, August 27 - September 3, 1995, Proceedings)*, volume 1043 of *Lecture Notes in Computer Science*, 1996. Springer. ISBN 3-540-60915-6. 98

M.O.Rabin and D.Scott. Finite automata and their decision problems. *IBM Journal of Research*, 3:115–125, 1959. 97

David E. Muller. Infinite sequences and finite machines. In *FOCS*, pages 3–16, 1963. 20

David E. Muller and Paul E. Schupp. Alternating automata on infinite trees. *Theor. Comput. Sci.*, 54:267–276, 1987. 125

David E. Muller and Paul E. Schupp. Simulating alternating tree automata by nondeterministic automata: new results and new proofs of the theorems of rabin, mcnaughton and safra. *Theor. Comput. Sci.*, 141(1&2):69–107, 1995. 126

David E. Muller, Ahmed Saoudi, and Paul E. Schupp. Alternating automata. the weak monadic theory of the tree, and its complexity. In *ICALP*, pages 275–283, 1986. 98, 99

David E. Muller, Ahmed Saoudi, and Paul E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *LICS*, pages 422–427, 1988. 126

J. S. Ostroff. *Temporal Logic for Real-Time Systems*. Research Studies Press, 1989. 40

Joël Ouaknine and James Worrell. On the decidability and complexity of metric temporal logic over finite words. *CoRR*, abs/cs/0702120, 2007. 34

Nir Piterman. Extending temporal logic with ω-automata. Master's thesis, The Weizmann Institute of Science, Israel, 2000. 101

Nir Piterman. From nondeterministic Büchi and streett automata to deterministic parity automata. *CoRR*, abs/0705.2205, 2007. 134, 137

Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. In *VMCAI*, pages 364–380, 2006. 118

Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977. 7, 24, 25, 46, 92, 116, 117

Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989. 32, 118, 121, 123, 125, 127, 128

Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In *Symposium on Programming*, pages 337–351, 1982. 92

Michael O. Rabin. *Decidability of second-order theories and automata on infinite trees*, volume 141. American Mathematical Society, 1969. 20, 125

Michael O. Rabin. Weakly definable relations and special automata. *Mathematical Logic and Foundations of Set Theory*, 59:1–23, 1970. 99

Michael O. Rabin. Automata on infinite objects and Church's problem. In *In Regional Conf.Ser.Math.*, pages 23–35, 1972. 32, 118

Jean-François Raskin. *Logics, Automata and Classical Theories for Deciding Real Time*. PhD thesis, FUNDP, Belgium, 1999. 140, 141

Jean-François Raskin and Pierre-Yves Schobbens. State clock logic: a decidable real-time logic. In *HART*, pages 33–47, 1997. 33

Jean-François Raskin and Pierre-Yves Schobbens. The logic of event clocks - decidability, complexity and expressiveness. *Journal of Automata, Languages and Combinatorics*, 4(3):247–286, 1999. 140, 141

Roni Rosner. *Applications and Optimizations for* LTL *Synthesis*. PhD thesis, Weizmann Institute of Science, Israel, 1992. 118

Harry Rudin. Network protocols and tools to help produce them. *Advances in Computers*, 2:291–316, 1987. 92

Shmuel Safra. On the complexity of ω -automata. In *FOCS*, pages 319–327, 1988. 9, 21, 95, 96, 98, 99, 109, 118, 125, 128

Sven Schewe and Bernd Finkbeiner. Bounded synthesis. In *ATVA*, pages 474–488, 2007. 9, 149

Fred B. Schneider, Bard Bloom, and Keith Marzullo. Putting time into proof outlines. In *REX Workshop*, pages 618–639, 1991. 41

D. Siefkesi. *Decidable Theories I- Büchi's Monadic second-order successor arithmetics*. Lecture Notes in Mathematics, Springer-Verlag, 1970. 96

A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985. 31

A. Prasad Sistla, Moshe Y. Vardi, and Pierre Wolper. The complementation problem for Büchi automata with appplications to temporal logic. *Theor. Comput. Sci.*, 49: 217–237, 1987. 97, 99

IEEE Computer Society. Ieee standard for a high performance serial bus. 1996. 42

Colin Stirling. *Handbook of Logic in Computer Science*. Oxford Science Publications, Clarendon Press, Oxford, 1992. 17

Robert S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control*, 54(1/2):121–141, 1982. 96, 126

H. Raymond Strong, Danny Dolev, and Flaviu Cristian. New latency bounds for atomic broadcast. In *IEEE Real-Time Systems Symposium*, pages 156–165, 1990. 41

Serdar Tasiran, Ramin Hojati, and Robert K. Brayton. Language containment of non-deterministic ω-automata. In *CHARME*, pages 261–277, 1995. 9, 95, 100, 126

James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968. 125

Boris A. Trakhtenbrot. *Finite automata and monadic second order logic*. Siberian Math.J, 1962. 125

Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Banff Higher Order Workshop*, pages 238–266, 1995. 30, 98

Moshe Y. Vardi. Reasoning about the past with two-way automata. In *ICALP*, pages 628–641, 1998. 126

Moshe Y. Vardi. Branching vs. linear time: final showdown. In *TACAS*, pages 1–22, 2001. 117

Moshe Y. Vardi and Larry J. Stockmeyer. Improved upper and lower bounds for modal logics of programs: preliminary report. In *STOC*, pages 240–251, 1985. 96

Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, pages 332–344, 1986. 31, 93, 127

Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Inf. Comput.*, 115(1):1–37, 1994. 31, 48, 93, 94

M.Y. Vardi. What makes modal logic so robustly decidable? *In Descriptive Complexity and Finite Models*, page 149183, 1997. 125

John von Neumann and Oskar Morgenstern. Theory of games and economic behavior. *Princeton University Press*, 1944. 115

Farn Wang. Parametric timing analysis for real-time systems. *Inf. Comput.*, 130(2): 131–150, 1996. 8, 39, 151

Farn Wang, Aloysius K. Mok, and E. Allen Emerson. Distributed real-time system specification and verification in APTL. *ACM Trans. Softw. Eng. Methodol.*, 2(4): 346–378, 1993. 41

Henri B. Weinberg and Lenore D. Zuck. Timed ethernet: real-time formal specification of ethernet. In *CONCUR*, pages 370–385, 1992. 41

Thomas Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In *FTRTFT*, pages 694–715, 1994. 33, 34

Martin Zimmermann. Parametric LTL games. Technical Report AIB 2010-20, RWTH Aachen University, 2010. 153, 154