Synthesis for a Weak Real-Time Logic

Omer Landry Nguena Timo

N° d'ordre : 3931

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

Par **Omer Landry Nguena Timo**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

SYNTHESE POUR UNE LOGIQUE TEMPS-REEL FAIBLE
(SYNTHESIS FOR A WEAK REAL-TIME LOGIC)

Soutenue le : 07 Décembre 2009

Après avis des rapporteurs :

Franck Cassez	Chargé de Recherche CNRS, HDR
François Laroussinie	Professeur, Université Paris Diderot – Paris 7

Devant la commission d'examen composée de :

Marc Zeitoun	Professeur, Université Bordeaux 1	Président du jury
Franck Cassez	Chargé de Recherche CNRS, HDR	Rapporteur
François Laroussinie	Professeur, Université Paris Diderot – Paris 7 ...	Rapporteur
Didier Lime	Maître de Conférences, École Centrale de Nantes	Examinateur
Igor Walukiewicz ...	Directeur de Recherche CNRS	Directeur de Thèse
André Arnold	Professeur à la retraite	Invité

– 2009 –

Remerciements

Synthèse Pour une Logique Temps-Réel Faible

Dans cette thèse, nous nous intéressons à la spécification et à la synthèse de contrôleurs des systèmes temps-réels. Les modèles pour ces systèmes sont des Event-recording Automata. Nous supposons que les contrôleurs observent tous les évènements se produisant dans le système et qu'ils peuvent interdirent uniquement des évènements contrôlables. Tous les évènements ne sont pas nécessairement contrôlables.

Une première étude est faite sur la logique Event-recording Logic (ERL). Nous proposons des nouveaux algorithmes pour les problèmes de vérification et de satisfaisabilité. Ces algorithmes présentent les similitudes entre les problèmes de décision cités ci-dessus et les problèmes de décision similaires étudiés dans le cadre du μ-calcul. Nos algorithmes corrigent aussi des algorithmes présents dans la litérature. Les similitudes relevées nous permettent de prouver l'équivalence entre les formules de ERL et les formules de ERL en forme normale disjonctive.

La logique ERL n'étant pas suffisamment expressive pour décrire certaines propriétés des systèmes, en particulier des propriétés des contrôleurs, nous introduisons une nouvelle logique WT_μ. La logique WT_μ est une extension temps-réel faible du μ-calcul. Nous proposons des algorithmes pour la vérification des systèmes lorsque les propriétés sont écrites en WT_μ. Nous identifions un fragment de WT_μ appelé WT_μ pour le contrôle (C-WT_μ). Nous proposons un algorithme qui permet de vérifier si une formule de C-WT_μ possède un modèle. Cet algorithme n'a pas besoin de connaître les ressources (horloges et constante maximale comparée avec les horloges) des modèles.

En utilisant C-WT_μ comme langage de spécification des systèmes, nous proposons des algorithmes de décision pour le contrôle centralisé et le Δ-contrôle centralisé. Ces algorithmes permettent aussi de construire des modèles de contrôleurs.

Mots-clés : Systèmes temps-réel, Event-Recording automata, logique temps-réel, satisfaisabilité, Event-Recording Logic, méthodes formelles, vérification, synthèse de contrôleurs.

Discipline : Informatique.

LaBRI,
Université Bordeaux 1,
351, cours de la Libération,
33405 Talence-Cedex (FRANCE).

Synthesis For a Weak Real-Time Logic

In this dissertation, we consider the specification and the controller synthesis problem for real-time systems. Our models for systems are kinds of Event-recording automata. We assume that controllers observe all the events occurring in the system and can prevent occurrences of controllable events.

We study Event-recording Logic (ERL). We propose new algorithms for the model-checking and the satisfiability problems of that logic. Our algorithms are similar to some algorithms proposed for the same problems in the setting of the standard μ-calculus. They also correct earlier proposed algorithms. We define disjunctive normal form formulas and we show that every formula is equivalent to a formula in disjunctive normal form.

Unfortunately, ERL is rather weak and can not describe some interesting real-time properties, in particular some important properties for controllers. We define a new logic that we call WT_μ. The logic WT_μ is a weak real-time extension of the standard μ-calculus. We present an algorithm for the model-checking problem of WT_μ. We consider a fragment of WT_μ called WT_μ for control (C-WT_μ). We show that the satisfiability of C-WT_μ is decidable. The algorithm that we propose for deciding whether a formula of C-WT_μ, has a model does not need to know the maximal constant used in models and it enables the construction of a witness model.

Using C-WT_μ, we present algorithms for a centralised controller synthesis problem and a centralised Δ-controller synthesis problems. The construction of witness controllers is effective.

Keywords: Real-time systems, Event-Recording automata, formal methods, real-time logic, μ-calculus, Event-Recording Logic, satisfiability, model-checking, controller synthesis.

Discipline: Computer Science.

LaBRI,
Université Bordeaux 1,
351, cours de la Libération,
33405 Talence-Cedex (FRANCE).

Contents

Introduction

A system is a set of interacting objects. We consider computerised systems, that are systems embedding computational devices. The role of computational devices is to execute programs. Computerised systems include transformational systems (classical systems whose inputs are available at the beginning of the execution, and which deliver their outputs when terminating: for instance compiler and batch systems), and *reactive systems* that react continuously to their environment, at the speed determined by the latter [HP85]. Reactive systems maintain an ongoing interaction with their environment while transformational systems do not. In their turn, reactive systems include interactive systems (for instance: World wide web browsers) and *real-time systems* (for instance control systems). While deadlines of computations, tasks or events can be occasionally missed in interactive systems, they should not be missed in real-time systems. Thus, the correctness of a real-time system not only depends on logical interactions that happen in it, but also on the time at which interactions (reception of inputs or outcomes of outputs) occur.

Because the size of computational devices have shrank, they are embedded in physical objects that they control. Control operations are often triggered by internal or external signals and the results of computations are used to introduce motion in physical objects of systems. In practice, the communication between computational devices and the rest of the system is realised using sensors that interpret physical input and actuators that introduce motions in the objects using results of computations. Compact disk players, mobiles phones, cars, washing machines and planes are examples of systems. For example there is around 30 program instructions in a washing machine and around one billion program instructions in a mobile phone.

According to the architecture of systems, it is usual to consider centralised systems and decentralised systems. *Centralised systems* embed a single computational device with a single program, while *decentralised systems* embed more than one computational device or program needing to communicate to achieve tasks including the control of physical devices.

The design of programs and systems should be a rigorous task and programs need to be validated. The validation approach that is widely considered consist to test the system with some test cases. The test cases are often generated manually by the developers. This validation approach is not reliable. Indeed, test cases could not cover all the aspects of the systems. For example, the test cases approach had not been efficient to discovered bugs in the phone switch CCS7 at Manathan(1990), in the phone communication network at Paris(1998), in the Ariane rocket(1999) [Fle02]. Just imagine a critical bug in the program that commands the exit of the wheels of an airplane or in systems embedded in automated cars.

On the other side, assume that one is able to detect bugs in a system and that one wants

to correct them. A solution may consist to correct erroneous parts of the system; if one is not able to correct the system, a solution may consist to start again the implementation of the system. Another solution may consist to combine the erroneous system with a new one in such a way that the resulting system does not longer contain bugs.

In this thesis we consider the correction problem for systems. The above motivates the use of formal methods presented below.

Formal Methods

It is often the case that requirements for systems are described in a natural language by customers, and implementation is performed by a team of engineers. It should be clear that, if the requirements are complex, so will also be the system. But, the simplicity of the requirements is not a guarantee for the simplicity and the correctness of the systems, as natural languages are often ambiguous. It is important to have *"exact"* languages to describe properties, *"exact"* methods to design systems, *"exact"* methods to validate or correct systems. These are the goals of *Formal methods* that include:

- *Test generation* aims at providing methods for asserting that a system is correct. It amounts to generation of a collection of test sequences from a formal specification and a property to be tested.

- *Proving correctness* amounts to providing a formal proof on the correctness of a system with respect to a property described in a formal language. This method is semi-automatic as it is often the case that prover needs human interaction (introduction of new axioms) to terminates.

- *Model-checking* is an automatic method that allows to check whether a system satisfies a given property.

- *Satisfiability/realisability* provides techniques to check whether a given property can be fulfilled by at least one system. It also provide techniques to construct a system that satisfies a given property.

- Controller synthesis designs *controllers* for a main system (called the *plant*) so that the *controlled system* satisfies a given property. It can be applied if the supervision of the system can be done by disabling in the plant some actions at the origin of the bugs.

All the classes above are somehow related. For example controller synthesis methods can be useful when a given system does not satisfy a property (test, proof, model-checking) and when the controllers can be constructed automatically (satisfiability).

We will consider the controller synthesis method for the correction of real-time systems when the properties are described with a "weak" real-time formal language. But, let us discuss some challenges concerning the definition of models for systems and the definition of formal languages to describe properties for systems.

The Design of models and Decision Procedures

The main goal of formal models is to provide formal representations for interesting "real-life" situations as we are not always interested in all the aspects of systems [HP85, Sif01].

The design of models for systems and properties depends on interesting aspects of systems and the nature of the properties for systems. Are we interested in the time at which a variation occurs in systems (if so, models may explicitly mention information on the time)? Are we interested only in the logical occurrence of events? Are we interested in a communication protocol (if so, a model may have a queue for message) or in reactive systems (if so, no queue is needed in the model). The design of models is based on the abstraction realised on interesting aspects of systems.

We are most often interested in the representation of behavioural aspects of systems. It is natural to think of behavioural aspects of systems as successive observable variations occurring in systems. Each variation may have a cause. Models for systems are abstraction of the variations and the causes of the variations. An abstraction of a variation contains: an abstraction of starting control point of the variation, an abstraction of the ending control point of the variation and, an abstraction of the cause of the variation. Abstract variations are often called *transitions*, abstract control points are often called *states* and abstract causes of variations are often called *events*.

To describe properties of systems, one can use natural languages; because they are ambiguous, they are neglected in favor of formal languages having exact semantics. The design of models for properties depends on the models for systems and the nature of properties for the models. Properties for models will combine properties on states and properties on transitions leading from a state (future-based properties) or on transitions leading to a state (past-based properties). The standard types of properties for systems include *reachability* properties (some situation can happen), *safety* properties (some situation will never happen), *liveliness* properties (some situation is unavoidable) *fairness* properties (some situation will happen infinitely often) *deadlock-free* properties (the system never stop).

Whatever is the nature and aspects of a system and the properties, it is obvious that their representations are useful in practice only if we are able to provide decision procedures for the validation and the correction problems including the model-checking, and the controller synthesis. The models of systems and the languages to describe properties are in this way, the result of an arbitration between the *expressive power* (that is class of systems and properties they can represent) , their *succinctness* enabling the representation of a big system with a model of small size, and their *simplicity* that makes their use easy and enables problems to become decidable (existence of decision procedures).

For practical issues, decision procedures need to be efficient and their implementation should be easy. For theoretical issues, it could happen that we are interested in the understanding of models and their theoretical properties. Then, we may not be interested in the efficiency of procedures, but only in relations with others decision procedures.

Abstraction of systems into models is the source of some problems including a *non determinism* of the models, *the (behavioral) equivalence of models* and the formalisation of the notion of combination of systems. A non determinism occurs in a model when two outgoing transitions from a state can be triggered by the same event; this can happen if an event is an abstraction of two different causes of variations from the same control point. Having two

models for the same system, knowing whether they are equivalent can reduce the complexity of decision procedures when a model is more tractable (for decision procedures) than the others. Models of systems can be combined in synchronous mode or asynchronous mode. In synchronous mode, a transition occurs in the combined system when from the respective current state of each component of the combination, it is possible to take a transition caused by a same event. In asynchronous model it is not required that the event happens at the same time in all the components. Reactive systems are often combined in synchronous mode.

Providing new languages for properties of systems raises fundamental problems of the language theory including emptiness checking (does a property have a model?), inclusion checking (is a set of models of a property included in a set of models of another property), the expressive power of languages (what properties can be described with a given language?) and closure properties under operations on languages (union, intersection, complementation). For example, for the closure under complementation, it could be useful and practical that the set of systems that do not satisfy a property can be described with another property written in the same language; for the closure under intersection, it could be useful that the conjunction of two properties can be described with a single property of the language.

Formal Models for Reactive Systems

A low level model for systems is *untimed transition system* that is just a collection of transitions. In that model, a transition $s \xrightarrow{a} s'$ indicates that in the state s, the process can move to the state s' when the event a occurs. No explicit information on the time of the occurrence of the events are mentioned. Untimed transition systems have been extended by adding a tripping condition on the transitions. For *probabilistic transition systems* [PZ93], tripping conditions are just probabilistic laws. For *timed transition systems* (TTS) [HMP92], tripping conditions are information on the time at which the event can occur. In timed transition systems, transitions are labelled either with a delay or by an event. Delay can range over a discrete domain (natural numbers for example) for *discrete time transition system* or over a dense domain (real numbers for example) for *dense time transition system*.

The problem with the low level models above is that they are not tractable for automation. They can not be represented using a finite structure as the set of states and the set of transitions in a model can be infinite. *High level models* that can be represented in a finite way have been developed.

There are two theoretical approaches for high-level modelling of systems. The algebraic approach [vG97] and the finite state transition systems based approach. The semantics of high level models is often described using low-levels models. Algebraic-based models can often be translated into transition systems. Thus, they will not be considered here. We describe below development that have been done for high-level models in the transition systems based approach.

Kripke structure or finite state automata [CCG00] are transition systems with finitely many states. Behaviours of the systems are a sequence of transitions.

Durational Kripke structure (DKS) [Lar05] are somehow a generalisation of ideas behind modeling systems with TTS or many other real-time models [EMSS91, CCG00, GHKK05].

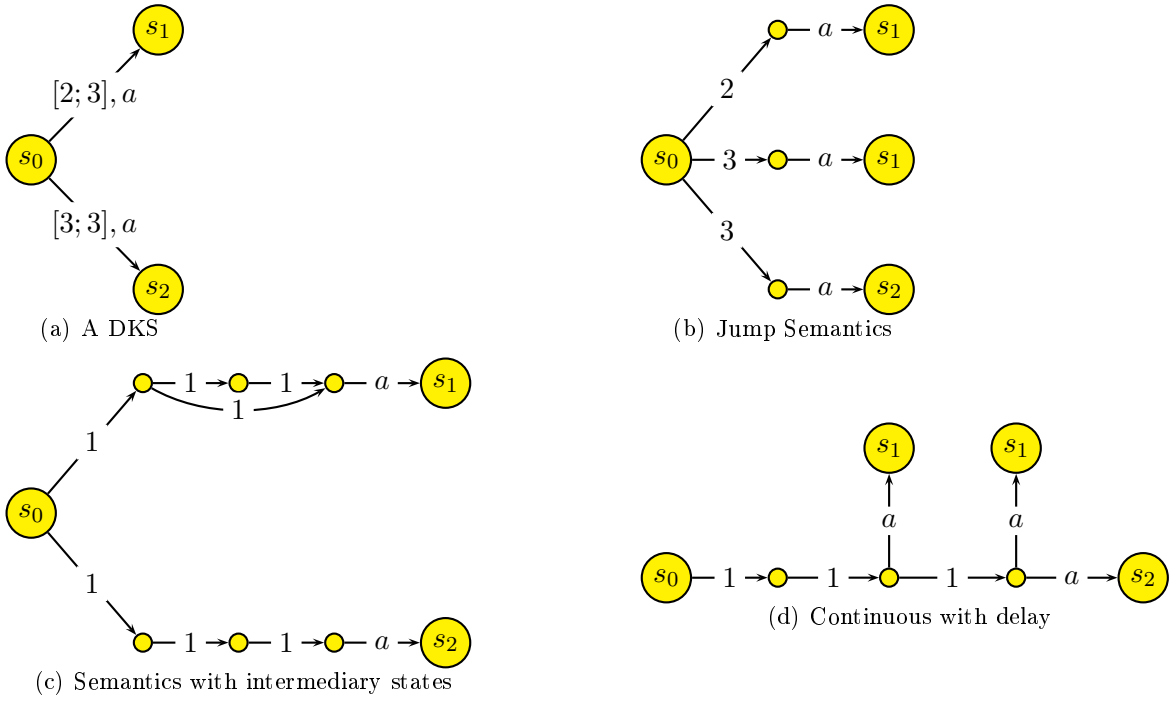

Figure 1: Example of a DKS and its semantics.

They are kinds of finite state automata having tripping conditions on transitions. There condition are integer interval. For example, a transition $s \xrightarrow{[2,3],a} s'$ asserts that the system can move from s to s' when if the event a occurs at 2, or 3 time units after the system enter the state s. Then it is fundamental to wonder about the states of the system 1, 2, or 3 times units after it enter the state s' or the state of the system between 1 and 2 times units. Three discrete semantics had been considered (see Figure 1 for illustration):

1. the *the jump semantics*. In this semantics the system move from a state s to s' without taking any intermediary state. Thus, the states of a system at the times $d+1$, $d+2$, ..., $d+t-1$ are not explicitly represented when moving from s in the time d to the state s' in the time $d+t$. Here time progresses but in a discrete way.

2. the *continuous semantics with intermediary states*. Here, moving from s to s' takes t time units and the system moves through intermediary states between s and s'. All the crossed intermediary states have the same properties as s. This semantics is not time deterministic but time elapse continuously. Then next reachable state is chosen very early in the time.

3. the *continuous semantics with delay*. The system lets the time elapse in its current state before moving to another state. This semantics is time deterministic and time elapses continuously. Moreover after an occurrence of an event, the next reachable state is chosen late in the time.

Timed automata [AD94] has been provided as powerful model to describe real-time systems. In timed automata, clock variables are used to handle the elapse of the time. All the

clock variables grow with the same rate (the one of the universal time). Timed automata are also a kind of timed extension of finite state automata. A transition $s \xrightarrow{g,a,X} s'$ is equipped with a tripping constraint g and a set of clock X to be reset when the transition is taken. The tripping constraint compares clock variables with rational constants. There is two types of comparisons: the comparison between a single clock and a rational called *diagonal free constraint* and the comparison of a difference between two clocks and a rational called *diagonal constraint*. Clocks are evaluated in the dense-time domain (set of positive real number). The *continuous semantics* of a high level transition $s \xrightarrow{g,a,X} s'$ is a set of low level transitions of the forms $(s,v) \xrightarrow{t} (s,v+t)$ or $(s,v) \xrightarrow{a} (s,v[X:=0])$ (v represents a valuation of the clocks, t represent a delay, $v+t$ returns the value of the clocks after the delay and $v[X:=0]$ return the values of the clocks after the reset of all the clock in X). A low level transition $(s,v) \xrightarrow{a} (s,v[X:=0])$ occurs only if the value of the clocks represented by v satisfies the condition g. We remark that since their introduction by Alur and Dill, timed automata have been extended to interesting models such as hybrid automata [Hen96], *updatable timed automata* [BDFP04].

Event-recording automata [AFH99] model has been introduced as a restricted form of timed automata. The main difference between Event-Recording automata and timed automata is that each clock is associated to a unique event and the unique clock to be reset when a transition is taken is the clock associated to the event on that transition.

Formal Description of Properties

Low level models are widely used to represent the semantics of high level models. Languages to described properties must be interpreted on transition systems models. Formal languages [Koy90, AH94, HR04] enable exact description of linear-time properties (that are properties on the possible executions) and/or branching-time properties (that are properties on states which may have several possible futures). They also describe either properties on untimed systems or properties on timed systems.

As it is discussed in [EH83], the use of linear or branching time specification languages depends on the underlying nature of time. The use of linear time language is based on the hypothesis that each moment (present time) has a unique future time while, when using branching time languages, we implicitly assume that the future of a present time could be divided into *alternative* future times. Alternation can be observed in non deterministic models as an event can be the source of two alternative transitions. Branching-time languages allow to characterize interesting behavioral relations between systems such as *simulation* and *bisimulation* [Par81]; they are sometimes preferred to linear-time languages.

The development of formal languages to describe properties has been done in two main directions: The logical direction that provides *logical languages* and the *automata-based* direction. We present here some relevant languages for untimed systems followed with some relevant languages for real-time (timed) systems.

The automata approach Automata, that are Kripke structures equipped with a set of accepting sequences of transitions, can be used to describe a system and its properties. A Kripke structure, describes the dynamics of a system. Behaviours (sequences of events or states) are

declared "good" or "bad" according to whether they belong to an acceptance condition (Rabin, Parity, etc...). Thus, automata [Tho90] are kinds of devices recognising set of words, set of trees, or set of transition systems. Automata on words are used to describe linear-time properties and automata on trees, in particular alternating automata on trees, are often used to describe branching-time properties.

Usual methods for emptiness checking and universality checking for automata-based languages consist to check whether some states are reachable. *Forward* or *Backward* state exploration algorithms are often used. The closure under boolean operations usually involves the construction of new automata.

We recall some results concerning some important problems on automata-based languages. Finite state automata on words and event-recording automata on timed words are closed under all boolean operations and the language inclusion testing problem is decidable. The language inclusion testing problem for these automata is also decidable. Timed automata on words are not closed under complementation and language inclusion testing problem is undecidable. The language emptiness testing problem for all the aforementioned automata on words is decidable. Alternating automata on trees (or transition systems) are closed under boolean operations, their emptiness and their inclusion testing problem is also decidable.

The logical approach Here are some important languages that have been developed.

Linear-temporal Logic (LTL) is a linear-time temporal logic introduced by Pnueli [Pnu77, LPZ85] for untimed systems. LTL enables the description of properties on a single execution of the system or sequence of transitions. But it is not possible to describe a property of the form "on all the executions of a system it is always true that there exists an execution that satisfies a property".

The *Computational Tree Logic* (CTL [CE82], CTL* [EH83]) are branching-time logics for untimed systems. They allow quantification (existential or universal) on transitions outgoing from states of models. Formulas of CTL* and CTL are not interpreted over independent set of executions (sequence of transitions) but over a tree-like structure representing a dependence between executions.

The *Hennessy-Milner logic* was introduced by Hennessy and Milner [HM80] for untimed systems (initially represented with the CCS algebraic representation). This logic include two important modal operators $\langle a \rangle$ and $[a]$. A state s of a system satisfies $\langle a \rangle \varphi$ when there is at least one outgoing transition $s \xrightarrow{a} s'$ such that the target s' satisfies the property φ. A state s of a system satisfies $[a]\varphi$ when for all outgoing transitions $s \xrightarrow{a} s'$, the target state s' satisfies the property φ. A weakness of the Hennessy-Milner logic is that it can not be used to described fairness or liveliness properties.

The *μ-calculus* introduced by Kozen [Koz82, AN01] is an expressive logic which extends the Hennessy-Milner Logic [HM80] by considering the greatest (ν) and least (μ) fixpoint operators. Fixpoint operators are useful to describe reachability, fairness or liveliness properties. The *μ-calculus* is more expressive than CTL and CTL*. Formulas of the μ-calculus includes arbitrary nested fixpoints. The power of nested fixpoint had been demonstrated [Bra98, BL05]; in particular they are used to describe properties such as: "an event occurs infinitely often" or "an event occurs almost all the time". The model-checking and the satisfiability problems of

the μ-calculus are decidable. The μ-calculus is as expressive as alternating automata on trees.

The metric temporal logic (MTL) [Koy90] is a timed linear-time temporal logic which extends LTL with timing constraints. For instance, with MTL, it is possible to write a formula expressing that a request p is always followed one time unit later by a response q. There are tractable fragments of MTL that had been considered like *Safety-MTL* [OW06b] which impose bound on the modality on the future and *MITL* [AFH96] which disallows punctual constraints some modality of the future.

Timed Computational Tree Logic (TCTL) [ACD93] is an extension of CTL that explicitly mentions the information on the time.

Timed Modal Logic(TML) and Extended Timed Modal Logic (ETML) are timed logic proposed by Larsen et al. [HLY91] to describe properties of processes expressed in the real-time process calculus TCCS of Wang [Yi90]. Among properties that can be handled with these logic an important property for real-time processes is the *necessity modal operator on time delays* that enables to describe a property like *"After a coin has been inserted, coffee will be continuously available for 30 seconds*. TML is an extension of the Hennessy-Milner Logic [HM80] which considers modalities of the form $\langle a\rangle_{\exists I}\varphi$, $\langle a\rangle_{\forall I}\varphi$, $[a]_{\exists I}\varphi$, and $[a]_{\forall I}\varphi$ where the semantics of I is a delay interval, a is an action and φ a formula and . A formula $\langle a\rangle_{\forall I}\varphi$ specifies a property which holds invariably for all time-delays in I; a system that satisfies this formula is such that any state reached after a time-delay within I must have a a-successor satisfying φ. A formula $\langle a\rangle_{\exists I}\varphi$ specify a property which holds eventually for some time-delay in I. Operator of the form $[a]_{\exists I}\varphi$ and $[a]_{\forall I}\varphi$ are defined by duality. Then, Larsen et al. have shown that if I is defined with a first-order assertion, then the model-checking of TML is decidable. ETML is a fragment of TML in which time intervals are not specified. In [HLY91] the model-checking and the satisfiability of ETML is left open.

The logic L^t_μ has been introduced by Sokolsky et al. [SS95]. The logic L^t_μ is a timed extension of the μ-calculus; it enables the description of safety and liveliness properties of real-time systems. The model-checking of L^t_μ is shown decidable in [SS95]. The logic L^t_μ supports all original operators of the μ-calculus as well as two new time modalities (necessity/universality and possibility/eventuality of time successors) also used in [HLY91]. But $L\mu^t$ formulas are *alternation free* as the fragment of the μ-calculus studied in [SS94, BC96], which means that in every L^t_μ formula the "level" of mutually recursive greatest and least fixpoint operators is one (arbitrary nested fixpoint is not authorized). The local model-checking algorithm for L^t_μ [SS95] uses quotients of clocks values as defined by Alur and Dill [AD94]. As this model-checking algorithm is local, the whole state space need not be explored and refinements of the quotient are carried only when necessary to satisfy clock constraints in the formula or the timed automaton used to represent the system under investigation.

The logic L_ν [LLW95] has been considered by Laroussinie et al.. It is a fragment of the logic T_ν [TXJS92] introduced by Henzinger et al.; it allows to describe properties on timed automata. Formulas of L_ν use also combined modalities on events of the classical μ-calculus with modalities on time-delays. The logic L_ν considers the greatest fixpoint operator; it does not consider the least fixpoint operator. The logic L_ν is sufficiently expressive for characterising timed automata (behavioral characterisation) [Cer93, SI94, IPPA00]. For a given timed automata, it is possible to construct a L_ν characteristic formula. The satisfiability problem of L_ν have been left open in [BCL05].

Event-Clock Logic (EventClockTL) has been proposed by Raskin and Schobbens [RS99].

EventClockTL is an extension of LTL with event-recording and event-predicting operators. The satisfiability problem of EventClockTL have been shown decidable.

Event-recording Logic (ERL) is a timed extension of the μ-calculus introduced by Sorea [Sor02] and it is used to describe properties on systems modelled with event-recording automata. ERL is more expressive than the event-recording part of EventClockTL since it includes arbitrary nested fixpoints. The extension consist in adding timing constraints in modal operators obtaining modal operator of the form $\langle g, a\rangle$ and $[g, a]$. For example, a formula of the form $\langle h_b < 3, a\rangle$ expresses the fact that the event a must occur at most 3 time units after the clock h_b has been reset (recall that the clock h_b is reset only after an occurrence of the event b). A decision procedure for the satisfiability problem of ERL is provided in [Sor02].

Let us comment some techniques used to solve some problems on logical languages. The closure properties are often a consequence of their definitions. Indeed most of the logical languages use boolean operators (logical "and" operator ($\wedge$) for the closure under intersection, logical "or" operator ($\vee$) for the closure under union, and logical "negation" operator ($\neg$) for the complementation). Duality is often a fundamental principle of logical languages.
The emptiness testing of logical languages is also called the *satisfiability problem*. A widely used method for temporal logic is the *tableau method*. Tableau systems were first developed by Gentzen as syntactical devices for modal logics [Gen34]. Tableau systems benefit from the structure of the properties to decompose their satisfiability checking into the satisfiability checking of smaller properties. It has been shown that there exists an intimate relationship between tableaux and automata over trees [Eme85].

Whatever is their forms (automata or logic), languages on the same models need to be compared. To compare two languages, it is common to provide example of properties that can be described with only one of the two languages and it is common to show how properties written in one of the two languages can be rewritten in the other language.

Methods and Algorithms for the Model-checking

Techniques for the model-checking of systems have been developed depending on models and the specifications. Most of these techniques work on low level models.

There are two basic strategies when designing a model-checking algorithm: *"Global"* algorithms that are recursive on the structure of the specification and evaluate each of part of the specification over the states of the transition system. *"Local"* algorithms, in contrast, explore only parts of the states space of the system, but check all parts of the specification. The choice of local or global algorithm does not affect the worst-case complexity of model-checking algorithms. Model-checking algorithms are often presented in the form of tableau [Eme85] and they use results on two player games.

In order to provide efficient model-checking algorithm, some techniques to reduce the size of models have been developed [Mer01] including, *symbolic techniques* and *abstraction based techniques*. *Symbolic techniques* consist in encoding set of states using compact objects such as logical formulas or efficient data structures [GV08] such as Binary Decision Diagrams, Difference Bound Matrices, Clock Difference Diagrams.

Let us recall some algorithms for the model-checking in some settings:

- The setting of Kripke Structures. The model-checking of systems modeled with Kripke structures has been widely investigated for linear-time properties (LTL [Var07]) and Branching-time Properties (CTL and CTL* [LS01], the μ-calculus [SE89]). The techniques aforementioned have been used and implemented in tools such as SMV [CCG$^+$02], MEC 5 [GV04], Lustre [CPHP87] and SPIN [Hol97].

- The setting of timed automata. Abstraction based techniques have been provided for the reachability of timed automata. These techniques include *region abstraction* and *zone-based abstraction*. They consist in partitioning the infinite set of states of the semantics into finite sets of abstraction classes. Region and zone-abstraction based techniques have also been deployed for other types of properties. This techniques have been used for the model-checking of TCTL [TXJS92], the model-checking of MTL and its fragments [OW05, OW06a, AH93, OW06b, AFH96], the model-checking of TML [HLY91], ETML [HLY91], L^t_μ [SS95], L_ν [LLW95] and Event-Recording Logic [Sor01, Sor02]. Tools implementing model-checking algorithms for real-time systems include Kronos [BDM$^+$98], Uppaal [BLL$^+$96], Hytech [HHWT97], Cmc [LL98], Tempo [Sor01] and PHAVer [Fre05, Fre08].

Methods and Algorithms for the Controller Synthesis

We recall that the supervisory control problem, as introduced by Ramadge and Wonham [RW89], asks whether a given system called *a plant* can be *controlled* with another system called a *controller* in such a way that the resulting system called the *controlled system* satisfies the given *control objective*. The synthesis problem asks whether a witness controller can be effectively computed.
The controller synthesis (control + synthesis) problem is studied depending on theoretical assumption made on the systems. These assumptions concern the nature of the events in systems, the architecture of systems and the nature of properties.

The notions of *controllability*, *observability*, *distinguishability* are often considered. The notion of controllability is based on the assumption that some events of the systems can be disable (controllable event) and the others can not. The notion of observability is based on the assumption that controller can not observe all the events that happen in the systems. This notion considers observable events and unobservable events. The notion of distinguishability relies on the fact that a controller may not abstract a cause of a variation in the same manner as the system; then, it could happen that occurrences of some events in the systems can not be distinguished by the controllers. Relying on the architecture of the systems, the *centralised supervisory control* is opposed to the *decentralised (distributed) control*. The centralised supervisory control of a system is achieved by a unique controller while in the decentralised cases, more than one controller can be combined with the plant to meet the control objectives. It is usual to distinguish *internal* control objectives from *external* control objectives. Internal control objectives refer to state properties while external control objectives refer to properties on sequences of events.

Let us recall some previous works on the controller synthesis for discrete event systems and dense-time systems.

Ramadge and Wonham [RW89] consider the supervisory control problem of discrete event

systems. In their setting, a plant and controllers are deterministic finite state automata; the notion of controllability is also considered. The external control objective is either a reachability or a safety property. Many authors [PR05, BK06, AVW03, AW07] have considered the supervisory control of discrete event systems modeled with finite state automata when the control objectives are described with a μ-calculus formula. These extensions of the works of Ramadge and Wonham use the expressive power of the μ-calculus to describe more general properties for supervised systems and controllers. They consider the notion of controllability, observability, and distinguishability and the centralised and decentralised supervisory control problems. They use a so-called *quotient based method* that provide powerful quotient operation for the division of properties by systems and the division of properties by properties. In the works of Arnold et al. [AVW03, ABPV05, AW07], the division operation works for *disjunctive normal form* formulas and the computation of witness controllers is effective (winning strategy in two player parity game) for some classes of decidable supervisory control problems. In particular, Arnold et al. [AW07] have shown that the supervisory control problem is decidable under the three following conditions: at most one controller is non deterministic; all but one specification of controllers are simple (a simple specification does not describe observability and distinguishability conditions); and the specification of the non deterministic controller is simple.

The centralised dense-time version of the supervisory control has been investigated and solved in [AMP95]. In that investigation, Maler et al. consider timed automata models, untimed control objectives and the notion of controllability. They provide an algorithm to decide whether a *discrete controller* exists, and show that if the answer is positive, a witness controller can be effectively computed. The control objective is a reachability or a safety property.
D'Souza and Madhusudan [DM02] have also considered the centralised dense-time supervisory control for timed automata when the external control objective is described with a timed automaton. They also consider the notion of controllability. For decidable cases of control, D'Souza and Madhusudan synthesise controllers with a *priory* limit on their resources (number of clocks, power of the controllers to observe clocks). Madhusudan et al. [BDMP03] had extended the frameworks of D'Souza and Madhusudan by considering the notion of partial observability.
Bouyer et al. [BBC06] have investigated centralised dense-time supervisory control of timed automata when the external control objective is described with the logic MTL. They consider notions of controllability and they provide decidability results when there is a limit on the resources of controllers. Controllers are just winning strategies in some two player parity games.
Laroussinie et al. [BCL05] have considered the centralised supervisory control problem for timed automata models with L_ν when the set of events is partitioned into a set of controllable events and a set of uncontrollable events. They present how to decide the existence of controller for some deterministic fragment of L_ν, but the procedure does not say how to construct a witness controller.

Contributions of this Thesis

We consider controller synthesis for real-time systems that can be combined in synchronous mode; the control objectives are timed branching-time properties.

The framework of Arnold et al. [AVW03, ABPV05, AW07], based on finite state automata and the μ-calculus, is a powerful framework for the controller synthesis of untimed systems. This work proposes methods to decide the existence of controllers and methods to synthesize controllers. For timed systems, Laroussinie et al. [BCL05] have provided an extension to the framework of Arnold et al. as they have considered timed automata models for systems and the logic L_ν to describe control objectives. The method in the Laroussinie et al. framework only decides the existence of a controller and does not provide a method to synthesise controllers.

Our goal in this thesis is to find a class of timed models "weaker" than the class of timed automata, to use a "weak" real-time extension of the μ-calculus for providing a powerful framework for the controller synthesis of real-time systems. We also hope to reuse techniques of the framework of Arnold et al.

We start our investigation with event-recording automata as models for systems and Event-Recording Logic (ERL) as language to describe properties. We present new decision procedures for the model-checking and the satisfiability of ERL. We also present a disjunctive normal form theorem for ERL. We show that ERL is not expressible enough to describe useful properties of timed processes, especially some interesting properties for controllers. For instance, with the modalities of ERL we are not able to describe a property of the form " an event can be completed at any moment that satisfies a timing constraint".

Then, we introduce a new logic that we call WT_μ which is also a "weak" real-time extension of the μ-calculus. We show that WT_μ is more expressive that ERL. We consider fundamental problems on WT_μ namely: the model-checking and the satisfiability problems. We show that the model-checking problem of WT_μ is decidable. For the satisfiability, we consider a fragment of WT_μ called C-WT_μ (WT_μ for the control). We provide a decision procedure for a satisfiability problem of C-WT_μ formulas. That procedure works without any information on the maximal constant of the models. It also shows how to construct a witness model for a satisfiable formula.

We present decision procedures for the centralised and the Δ-dense-time centralised controller synthesis problems when the control objectives are described with C-WT_μ formulas.

Organisation of this Thesis

In Chapter 1 we present basic notions that we use later in the thesis. These notions include alternating automata on trees, two player parity games, the μ-calculus, the logic L_μ and some frameworks to the controller synthesis.

In Chapter 2, we present models for real-time systems and some fundamental problems about these models. Our model, that we call *timed process* is nothing else but event-recording automata (without an acceptance condition). We present the reachability analysis in that model using well know region abstraction techniques and zone abstraction technique. We present how to remove diagonal constraints in the model without changing their behavioural properties.

In Chapter 3, we present Event-Recording Logic (ERL for short). We consider fundamental problems about that logic: the model-checking problem, the satisfiability problem, the disjunctive normal form problem. The first two problems have been considered earlier by Sorea [Sor02]. Our algorithms for these problems enable a better understanding of the models;

they also enable to reuse some algorithms for the same problems for the standard μ-calculus. We provide a disjunctive normal form theorem for ERL formulas. We show that the algorithm of Sorea [Sor02] for the satisfiability checking is ambiguous and is not correct in case of diagonal constraints.

The Chapter 4 introduces the new logic WT_μ. There, we define WT_μ and we show that WT_μ is more expressive than ERL as any formula of ERL can be translated into equivalent formula of WT_μ and some formulas of WT_μ can not be translated into formulas of ERL. WT_μ enables a description of some interesting properties in particular some properties of controllers. Then we consider the model-checking and the satisfiability problems for WT_μ. We show that the model-checking of WT_μ is decidable. We introduce C-WT_μ as a decidable fragment of WT_μ. Our decision procedure for the satisfiability of C-WT_μ shows how to construct models for satisfiable formulas.

The centralised and the Δ-dense time centralised controller synthesis problems are considered in Chapter 5. Formulas are difficult to handle because they use fixpoint operators. We introduce modal automata that are a kind of alternating automata. Modal automata are interpreted over timed processes. We define the quotient of modal automata over timed processes. Then, we consider a subclass of modal automata that we call modal automata for control (C-MA). We show that a C-MA automaton can be translated into an equivalent C-WT_μ formula and reciprocally a C-WT_μ formula can be translated into an equivalent C-MA automaton. At the end of this chapter, we show that the two aforementioned controller synthesis problems are decidable; moreover we show how to construct controllers.

Chapter 1

Preliminaries

This chapter presents some frameworks for the supervisory control problem of discrete systems and dense-time systems.

First of all, we recall some definitions and results concerning transition systems, automata on words, the standard μ-calculus and automata on transition systems. Then, we present the frameworks of Ramadage and Wonham [RW89], Arnold et al. [AVW03, ABPV05, AW07], D'Souza and Madhusudan [DM02] and Laroussinie et al. [BCL05].

1.1 Automata on Words

In this section we present basic notions that include labelled transition systems, bisimulation relation, (Büchi, Rabin and Parity) acceptance conditions and automata on words.

Definition 1 A *word* over an alphabet Σ is a sequence $w = w_0.w_1 \ldots$ of symbols in Σ. Σ^* is the set of finite words over Σ, and Σ^ω is the set of infinite words over Σ.

For a word w, the number of occurrences of the letter a in w is denoted by $|w|_a$. Given $w \in \Sigma^\omega$, we consider the set

$$Inf(w) = \{a \in \Sigma \,|\, \forall i \exists j > i \, w_j = a\}$$

of symbols in Σ occurring infinitely often in w.

Definition 2 A *labelled transition system* over an alphabet Σ (or a Σ-*labelled transition system* for short) is a tuple $\mathcal{S} = \langle S, \Sigma, s^0, \Delta_\mathcal{S} \rangle$ where S is a set of states, $s^0 \in S$ is the initial state and $\Delta_\mathcal{S} \subseteq S \times \Sigma \times S$ is a transition relation. A Σ-labelled transition system is *deterministic* if $\Delta_\mathcal{S}$ is a partial function $\Delta_\mathcal{S} : S \times \Sigma \to S$.

We often write $s \xrightarrow{a} s'$ instead of simply the transition $(s, a, s') \in \Delta_\mathcal{S}$.

A *finite* labelled transition system is a system with finitely many states.

Definition 3 A *product* of two Σ-labelled transition systems $\mathcal{P} = \langle P, \Sigma, p^0, \Delta_P \rangle$ and $\mathcal{S} = \langle S, \Sigma, s^0, \Delta_\mathcal{S} \rangle$ is the Σ-labelled transition system $\mathcal{P} \times \mathcal{S} = \langle P \times S, \Sigma, (p^0, s^0), \Delta_{\mathcal{P}\times\mathcal{S}} \rangle$ where $(p, s) \xrightarrow{a} (p', s')$ if and only if $p \xrightarrow{a} p'$ and $s \xrightarrow{a} s'$.

We define two behavioral relations between label transition systems. These relations, called simulation and bisimulation, have been introduced by Park [Par81]. Let $\mathcal{S}_1 = \langle S_1, \Sigma, s_1^0, \Delta_{\mathcal{S}1} \rangle$ and $\mathcal{S}_2 = \langle S_2, \Sigma, s_2^0, \Delta_{\mathcal{S}2} \rangle$ be two labelled transition systems.

Definition 4 A *simulation between* $\mathcal{S}_1$ *and* $\mathcal{S}_2$ is a relation $\mathcal{R} \subseteq S_1 \times S_2$ such that whenever $s_1 \mathcal{R} s_2$ and $a \in \Sigma$, then:

- If $s_1 \xrightarrow{a} s_1'$ then there exists $s_2' \in S_2$ such that $s_2 \xrightarrow{a} s_2'$ and $s_1' \mathcal{R} s_2'$.

Definition 5 A *bisimulation between* $\mathcal{S}_1$ *and* $\mathcal{S}_2$ is a relation $\mathcal{R} \subseteq S_1 \times S_2$ such that whenever $s_1 \mathcal{R} s_2$ and $a \in \Sigma$, then:

- If $s_1 \xrightarrow{a} s_1'$ then there exists $s_2' \in S_2$ such that $s_2 \xrightarrow{a} s_2'$ and $s_1' \mathcal{R} s_2'$.
- If $s_2 \xrightarrow{a} s_2'$ then there exists $s_1' \in S_1$ such that $s_1 \xrightarrow{a} s_1'$ and $s_1' \mathcal{R} s_2'$.

We write $s_1 \sqsubseteq s_2$ (resp. $s_1 \sim s_2$) if and only if there exists a simulation (resp. a bisimulation) $\mathcal{R}$ with $s_1 \mathcal{R} s_2$.

Definition 6 $\mathcal{S}_2$ *simulate* $\mathcal{S}_1$ (resp. $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *are bisimilar*) whenever there exists a simulation (resp. a bisimulation) $\mathcal{R}$ between $\mathcal{S}_1$ and $\mathcal{S}_2$ such that the pair (s_1^0, s_2^0) of their initial states belongs to the relation $\mathcal{R}$, and then we write $\mathcal{S}_1 \sqsubseteq \mathcal{S}_2$ (resp. $\mathcal{S}_1 \sim \mathcal{S}_2$).

Definition 7 An *ω-automaton* on words over Σ is a tuple $\mathcal{A} = \langle \mathcal{S}, Acc \rangle$ where $\mathcal{S} = \langle S, \Sigma, s^0, \Delta_{\mathcal{S}} \rangle$ is a finite Σ-labelled transition system and Acc is the *acceptance condition*.

Definition 8 Let $\mathcal{A} = \langle \mathcal{S}, Acc \rangle$ be an ω-automaton over Σ-words as above defined. A *run* ρ of $\mathcal{A}$ on a word $w = w_0 w_1 \cdots \in \Sigma^\omega$ is a sequence of states $\rho = s_0 s_1 \ldots$ such that the following conditions hold:

1. $s_0 = s^0$
2. s_i is such that $s_{i-1} \xrightarrow{w_i} s_i \in \Delta_{\mathcal{S}}$

Whether a run of an automaton is accepting depends on the nature of the acceptance condition of the automaton. There are several acceptance conditions:

1. *The Büchi acceptance condition* [Bö62] is given by a set $F \subseteq Q$: ρ is accepting when
$$Inf(\rho) \cap F \neq \emptyset$$
2. *The Rabin acceptance condition* [Rab69] is given by a set $\Omega = \{(E_i, F_i)\}_{i=1..n}$ with $E_i, F_i \subseteq Q$: ρ is accepting when
$$\exists (E, F) \in \Omega \, s.t \, (Inf(\rho) \cap E = \emptyset) \wedge (Inf(\rho) \cap F \neq \emptyset)$$

3. *The parity condition* [Mos85] is given by a function $rank : Q \to \{1, \ldots, k\}$ (where k is a natural number) that assigns a *parity index* to states of the automaton: ρ is accepting when
$$max\{rank(q) \mid q \in Inf(\rho)\}$$
is even. This condition is also called the *max-parity condition.*

Depending of the nature of the acceptance condition, automata are called Büchi automata, Rabin automata, or Parity automata.

Definition 9 The *language* of an automaton $\mathcal{A}$ denoted by $\mathcal{L}(\mathcal{A})$ is the set of words on which $\mathcal{A}$ has an accepting run.

Let us recall some interesting well known results on automata. Non deterministic Büchi automata, Rabin automata and Parity automata accept the same set of languages. This set of languages is closed under intersection, union, and complementation (see [Tho97]). The emptiness checking for a Rabin automata with m states and n pairs is decidable in $O(mn)^{3n}$. Every non deterministic Rabin automaton can be translated into an equivalent parity automaton and reciprocally (see [L$\ddot{9}$9]). Moreover, every non deterministic parity automaton can be translated into a deterministic parity automaton.

1.2 Two Player Parity Games and Multi-Parity Games

We present a complexity result for checking a winning strategy in a two player games with parity condition. We also present the notion of two multi-parity game.

Definition 10 A *two player parity game*(see [Zie98]) is a tuple $\mathcal{G} = \langle N_E, N_A, T \subseteq N^2, Acc_{\mathcal{G}} \rangle$ where $\langle N, T \rangle$ is a graph with the nodes (or positions) $N = N_A \cup N_E$ partitioned into N_E and N_A. N_E denotes the set of nodes of the player *Eve* and N_A denotes the set of nodes of the player *Adam*. The winning condition $Acc_{\mathcal{G}} \subseteq N^\omega$, is a parity condition on the nodes. The game is finite if N is finite.

A *play* between *Eve* and *Adam* from some node $n \in N$ proceeds as follows: if $n \in N_E$ then *Eve* makes a choice of a successor otherwise *Adam* chooses a successor; from this successor the same rule applies and the play goes on forever unless one of the parties cannot make a move. A play is finite if a player cannot make a move and then he loose the play. In the case that the play is an infinite path $\pi = n_0 n_1 n_2 \cdots$, *Eve* wins if $\pi \in Acc_{\mathcal{G}}$. Otherwise *Adam* is the winner. Among winning conditions introduced in the literature, we consider the parity condition. A *strategy* σ for *Eve* is a function assigning to every sequence of nodes $\vec{n}$ ending in a node n from N_E a vertex $\sigma(\vec{n})$ which is a successor of n.

A *play from* n *consistent* with σ is a finite or infinite sequence $n_0 n_1 n_2 \cdots$ such that $n_{i+1} = \sigma(n_i)$ for all i with $n_i \in N_E$. The *strategy* σ *is winning for Eve* from the node n if and only if all the plays starting in n and consistent with σ are winning. The strategies for *Adam* is are defined similarly. A *node is winning* if there exists a strategy winning from it. A game is *determined* if every node is winning for one of the player. A strategy is *positional* if it does not depend on the sequences of nodes that were played till now, but only on the present node.

So such a strategy for *Eve* can be represented as a function $\sigma : N_E \to N$ and identified with a choice of edges in the graph of the game.

Now we state the following results on two player games (see [GH82, EJ91, Jur00, VJ00]).

Theorem 11 *Every parity game is determined. In a two player parity game one of the players has a winning positional strategy from each of his winning nodes. There is an effective procedure that decides who is a winner from a given node in a finite game, and that procedure works in time*

$$\mathcal{O}\left(|T| \times \left(\frac{2 \times |N|}{d}\right)^{\lceil d/2 \rceil}\right)$$

where, d is the maximal parity index.

1.3 The μ-Calculus

The μ-calculus introduced by Kozen [Koz82] (see also [AN01]) is an expressive temporal logic that extends modal logic with the greatest (ν) and least (μ) fixpoint operators. We present the syntax and the semantics of the μ-calculus. Then we state some well known results that include the complexity of the model-checking problem, the complexity of the satisfiability problem and a disjunctive normal form theorem. The complexity result for the model-checking is obtained by reduction to checking if there is a winning strategy in a two player parity game.

1.3.1 Definitions and Semantics

Definition 12 The syntax of the μ-calculus is defined over a set $Var = \{X, Y, \ldots\}$ of variables, a set Σ of events. It is given by the following grammar:

$$\varphi ::= tt \,|\, ff \,|\, X \,|\, \varphi \vee \psi \,|\, \varphi \wedge \psi \,|\, \langle a \rangle \varphi \,|\, [a]\varphi \,|\, \mu X.\varphi(X) \,|\, \nu X.\varphi(X)$$

In the above, $X \in Var$, $a \in \Sigma$; and tt and ff denote the formula that are always "true" and "false" respectively; $\langle a \rangle$ and $[a]$ denote the existential and the universal modalities indexed with the event a; they represent "exists a-successor and "all a-successor" modalities respectively. The formulas $\mu X.\varphi(X)$ and $\nu X.\varphi(X)$ represent respectively the least and the greatest fixpoint formula.

For a formula φ, the closure [Koz82] of φ, $sub(\varphi)$ is defined as follows:

Definition 13 The *closure* $sub(\varphi)$ of φ is the smallest set of formulas such that:

- $\varphi \in sub(\varphi)$
- if $\psi_1 \vee \psi_2 \in sub(\varphi)$ the both $\psi_1, \psi_2 \in sub(\varphi)$
- if $\psi_1 \wedge \psi_2 \in sub(\varphi)$ the both $\psi_1, \psi_2 \in sub(\varphi)$
- if $\langle a \rangle \psi \in sub(\varphi)$ then $\psi \in sub(\varphi)$
- if $[a]\psi \in sub(\varphi)$ then $\psi \in sub(\varphi)$

- if $\sigma X.\psi(X) \in sub(\varphi)$ then $\psi(X) \in sub(\varphi)$, where $\sigma \in \{\nu, \mu\}$

The formulas in $sub(\varphi)$ are called the *sub formulas* of φ. For a formula φ, $sub(\varphi)$ is finite and, by definition, it is not larger that the number of symbols used in φ.

Definition 14 The set $free(\varphi)$ of free variable of a μ-calculus formula φ is defined inductively as follows:

- $free(tt) = free(ff) = \emptyset$
- $free(X) = \{X\}$
- $free(\varphi \vee \psi) = free(\varphi \wedge \psi) = free(\varphi) \cup free(\psi)$
- $free([a]\varphi) = free(\langle a \rangle \varphi) = free(\varphi)$
- $free(\mu X.\varphi(X)) = free(\nu X.\varphi(X)) = free(\varphi) \setminus \{X\}$

A variable X is *free* in a formula φ if $X \in free(\varphi)$.

Definition 15 A variable X is *bound* in a formula φ if there is a sub formula $\sigma X.\psi(X)$ of φ with $\sigma \in \{\mu, \nu\}$.

We remark that, a variable can be bound and free at the same time. For example, in the formula $\varphi = \mu X(\langle a \rangle X \vee \langle b \rangle Y) \wedge \nu Y.\langle c \rangle Y$, the variable Y is bound and free. An occurrence of Y in φ can be replaced with a new variable to get an equivalent formula variables of which are either free or bound.

Definition 16 (Well named) We call a formula *well named* if the expression $\mu X.\varphi(X)$ (or $\nu X.\varphi(X)$) occurs at most once for each variable X.

By renaming some occurrences of variables if necessary, every formula can be translated into an equivalent well named formula. In what follows, without loss of generality, we assume that formulas are well named.

Definition 17 (Binding) The *binding definition* of a bound variable X in a well named formula φ, $\mathcal{D}_\varphi(X)$ is the unique sub formula of φ of the form $\sigma X.\psi(X)$. We will omit subscript φ when it causes no ambiguity. We call X a *μ-variable* when $\sigma = \mu$, otherwise we call X a *ν-variable*. The function $\mathcal{D}_\varphi$ assigning to every bound variable its binding definition in φ will be called the *binding function* associated with φ.

Definition 18 A *sentence* is a well named formula without free variables.

Definition 19 The *dependency order* $\leq_\varphi$ over the bound variables of a formula φ, is the least partial order such that if X occurs in $\mathcal{D}_\varphi(Y)$ and $\mathcal{D}_\varphi(Y)$ is a sub formula of $\mathcal{D}_\varphi(X)$ then $X \leq_\varphi Y$. When $X \leq_\varphi Y$, it is also said that Y *depends on* X or X is *older than* Y.

Let us illustrate the three definitions just above with an example. Consider again the formula $\varphi = \mu X(\langle a\rangle X \vee \langle b\rangle Y) \wedge \nu Y.\langle c\rangle Y$. It should be clear that φ is not a sentence as there is a free occurrence of the variable Y in φ. We have that $\mathcal{D}_\varphi(X) = \mu X(\langle a\rangle X \vee \langle b\rangle Y)$ and $\mathcal{D}_\varphi(Y) = \nu Y.\langle c\rangle Y$. The variables X and Y can not be compared with the dependency order relation $\leq_\varphi$.

Definition 20 (Expansion) Given a formula φ, its binding function $\mathcal{D}_\varphi$, and a sub formula ψ of φ, the *expansion* $\langle\!|\psi|\!\rangle_{\mathcal{D}_\varphi}$ of ψ with respect to $\mathcal{D}_\varphi$ is defined by

$$\langle\!|\psi|\!\rangle_{\mathcal{D}_\varphi} = \psi[\mathcal{D}_\varphi(X_n)/X_n]\cdots[\mathcal{D}_\varphi(X_1)/X_1]$$

where $X_1 \leq_\varphi X_2 \leq_\varphi \cdots \leq_\varphi X_n$ is a chain of bound variables of φ with respect to $\leq_\varphi$.

Definition 21 Variable X in $\mu X.\varphi(X)$ is *guarded* if every occurrence of X in $\varphi(X)$ is in the scope of some modality operator $\langle\rangle$ or $[]$. We say that a *formula is guarded* if every bound variable in the formula is guarded.

Alternation depth describes the number of alternations between least and greatest fixpoint operators.

Definition 22 The *alternation depth* of a formula denoted by $alt(\varphi)$ is the number of nesting between μ and ν in φ; it is recursively defined as follows:

- $alt(tt) = alt(ff) = alt(X) = 0$
- $alt(\varphi \wedge \psi) = alt(\varphi \vee \psi) = \max(alt(\varphi), alt(\psi))$
- $alt(\langle a\rangle\varphi) = alt([a]\varphi) = alt(\varphi)$
- $alt(\mu X.\varphi(X)) = \max(\{1, alt(\varphi(X)\} \cup \{1 + alt(\nu Y.\psi(Y)) \,|\, \nu Y.\psi(Y) \in sub(\varphi); X \leq_\varphi Y\})$
- $alt(\nu X.\varphi(X)) = \max(\{1, alt(\varphi(X)\} \cup \{1 + alt(\mu Y.\psi(Y)) \,|\, \mu Y.\psi(Y) \in sub(\varphi); X \leq_\varphi Y\})$

Formulas of the μ-calculus are interpreted over Σ-labelled transition systems. The semantics of a μ-calculus formula φ is a set of states of a Σ-labelled transition system $\mathcal{S} = \langle S, \Sigma, s^0, \Delta_\mathcal{S}\rangle$ where the formula holds under a given valuation of variables $Val : Var \to 2^S$, and it is denoted by $[\![\varphi]\!]^\mathcal{S}_{Val}$. Given a valuation of variables Val and a set of states $T \subseteq S$, the valuation $Val[X/T]$ is the valuation Val with the substitution that associates the states of T with the variable X. Formally, for $Y \in Var$, $Val[X/T](Y) = T$ if $Y = X$ and $Val(Y)$ otherwise. We define the relation $\vDash$ between a state s of a transition system $\mathcal{S}$, a valuation Val and a formula φ. We write $\mathcal{S}, s, Val \vDash \varphi$ when the formula φ holds in s or equivalently s satisfies φ. The relation $\vDash$ is defined as follows:

- $\mathcal{S}, s, Val \vDash X$ if $s \in Val(X)$
- $\mathcal{S}, s, Val \vDash \varphi_1 \vee \varphi_2$ if $\mathcal{S}, s, Val \vDash \varphi_1$ or $\mathcal{S}, s, Val \vDash \varphi_2$
- $\mathcal{S}, s, Val \vDash \varphi_1 \wedge \varphi_2$ if $\mathcal{S}, s, Val \vDash \varphi_1$ and $\mathcal{S}, s, Val \vDash \varphi_2$
- $\mathcal{S}, s, Val \vDash \langle a\rangle\varphi$ if there is $s \xrightarrow{a} s'$ such that $\mathcal{S}, s', Val \vDash \varphi$

- $\mathcal{S}, s, Val \models [a]\varphi$ if for all $s \xrightarrow{a} s'$ we have $\mathcal{S}, s', Val \models \varphi$
- $\mathcal{S}, s, Val \models \mu X.\varphi(X)$ if $s \in \bigcap\{T \subseteq S \mid [\![\varphi(X)]\!]^{\mathcal{S}}_{Val[X/T]} \subseteq T\}$.
- $\mathcal{S}, s, Val \models \nu X.\varphi(X)$ if $s \in \bigcup\{T \subseteq S \mid T \subseteq [\![\varphi(X)]\!]^{\mathcal{S}}_{Val[X/T]}\}$

Then we define $[\![\varphi]\!]^{\mathcal{S}}_{Val} = \{s \in \mathcal{S} \mid \mathcal{S}, s, Val \models \varphi\}$. It is said that a Σ-labelled transition system $\mathcal{S}$ is a *model* of a formula φ when $s^0 \in [\![\varphi]\!]^{\mathcal{S}}_{Val}$; in this case we write $\mathcal{S}, Val \models \varphi$. The valuation Val is omitted if the formula does not contains free variables.

It is known (see [Eme90] for a survey) that properties expressed in temporal logics LTL, CTL, and CTL* can be encoded as μ-calculus formulas and that there are formulas of the μ-calculus (for instance $\nu X.\langle a\rangle\langle a\rangle X$) that can not be written in CTL*.

Given two formulas φ_1 and φ_2, we often use the notation $\varphi_1 \equiv \varphi_2$ to say that φ_1 is *equivalent* to φ_2, meaning that for every labelled transition system $\mathcal{S}$ and valuation Val, $[\![\varphi_1]\!]^{\mathcal{S}}_{Val} = [\![\varphi_2]\!]^{\mathcal{S}}_{Val}$.

It is standard to consider the negation operator ($\neg$) on μ-calculus sentences. Given a formula φ and a Σ-labelled transition system $\mathcal{S}$ and a valuation Val, this operator is defined by: $[\![\neg\varphi]\!]^{\mathcal{S}} = S \setminus [\![\varphi]\!]^{\mathcal{S}}$.

The following proposition is standard.

Proposition 23 The following equivalences are true:

- $\neg tt \equiv ff$
- $\neg ff \equiv tt$
- $\neg(\varphi_1 \wedge \varphi_2) \equiv \neg\varphi_1 \vee \neg\varphi_2$
- $\neg(\varphi_1 \vee \varphi_2) \equiv \neg\varphi_1 \wedge \neg\varphi_2$
- $\neg\langle a\rangle\varphi \equiv [a]\neg\varphi$
- $\neg[a]\varphi \equiv \langle a\rangle\neg\varphi$
- $\neg\mu X.\varphi(X) \equiv \nu X.\neg\varphi(\neg X)$
- $\neg\nu X.\varphi(X) \equiv \mu X.\neg\varphi(\neg X)$

Thanks to the proposition just above, the negation operator can not appear in μ-calculus sentences.

Let us present some results on the μ-calculus.

Proposition 24 ([Koz82]) Every formula of the μ-calculus is equivalent to some guarded formula of the μ-calculus.

1.3.2 Model-Checking and Satisfiability Results

Informally, the task of checking whether a finite state transition system, $\mathcal{S} = \langle S, \Sigma, s^0, \Delta_{\mathcal{S}} \rangle$ is a model of a sentence φ can be seen as two player parity game whose nodes are set of tuples of the form (s, ψ) where $s \in S$ and ψ is a sub formula of φ. Positions of the player *Eve* contain sub formulas of one of the forms $tt, \varphi_1 \vee \varphi_2, \langle a \rangle \psi$. The other positions belong to the player *Adam*. The initial position of the game is (s^0, φ). The set of moves of the games are such that:

- There is no move from either (s, tt) or (s, ff).
- From $(s, \varphi \wedge \psi)$ as well as from $(s, \varphi \vee \psi)$ there are moves to (s, φ) and to (s, ψ).
- From $(s, [a]\varphi)$ and from $(s, \langle a \rangle \varphi)$ there are moves to $(s', \varphi$, for every s' such that $s \xrightarrow{a} s'$.
- There is a move from $(s, \sigma X.\varphi(X))$ to $(s, \varphi(X))$
- There is a move from X to $(s, \varphi(X))$ where $\mathcal{D}(X) = \sigma X.\varphi(X)$

The acceptance condition is given by the parity function $rank : Q \to \mathbb{N}$ defined by:

$$rank(\psi) = \begin{cases} 0 & \text{if } \psi \text{ is not a variable} \\ 2 \times alt(\mathcal{D}(X)) & \text{where } \varphi = X \text{ and } X \text{ is a } \nu\text{-variable} \\ 2 \times alt(\mathcal{D}(X)) + 1 & \text{where } \varphi = X \text{ and } X \text{ is a } \mu\text{-variable} \end{cases}$$

One can show that S is a model of a formula if player *Eve* has a winning strategy in the the game. This gives an intuitive idea behind the following results.

Theorem 25 ([EJ91, Tho97, Jur00]) *Let $\mathcal{S} = \langle S, \Sigma, s^0, \Delta_{\mathcal{S}} \rangle$ be a Σ-labelled transition system and let φ be a μ-calculus formula. The model-checking problem for φ and $\mathcal{S}$ is solvable in time*

$$\mathcal{O}\left(|\Delta_{\mathcal{S}}| \times |sub(\varphi)| \times \left(\frac{|S| \times |sub(\varphi)|}{\lfloor alt(\varphi)/2 \rfloor}^{\lceil alt(\varphi)/2 \rceil}\right)\right)$$

Theorem 26 ([Cas87, AD89, Sti96]) *Let $\mathcal{S}_1$ and $\mathcal{S}_2$ two bisimilar labelled transition systems. For every μ-calculus formula φ, $\mathcal{S}_1$ is a model of φ is and only if $\mathcal{S}_2$ is a model of φ.*

Theorem 27 ([EJ91]) *The satisfiability problem for μ-calculus formulas is solvable in exponential time. The construction of witness models is effective.*

In [EJ91, NW96], two player parity games are defined for the satisfiability problem of the μ-calculus. The authors shown that a μ-calculus formula is satisfiable if and only if there is a winning strategy for one of the two players in the game. A witness model for a formula is presented as a winning strategy. We recall that two player parity games are determined and strategies are positional. It follows that witness models for formulas are finite state automata.

1.3.3 Disjunctive Normal Form

Disjunctive normal form formulas are special kinds of formulas. One restriction in disjunctive normal form formulas is that conjunctions of the form $\langle a\rangle\varphi \wedge \langle a\rangle\psi$ are not allowed. These conjunctions are in some sense deterministic as for example, there is not need to check whether a state of a transition system satisfies two formulas after the occurrence of an event; and a model for a formula φ can be "easily" merged with a model of a formula ψ in order to build a model for $\langle a\rangle\varphi \wedge \langle b\rangle\psi$. The other restrictions are presented in Definition 29.
Let us present a modal operator [JW95] that extends the syntax of the μ-calculus.

Definition 28 Let Γ be a set of formula. The operator $(a) \rightarrow\Gamma$ is defined by

$$(a) \rightarrow\Gamma = \bigwedge_{\varphi\in\Gamma} \langle a\rangle\varphi \wedge [a] \bigvee_{\varphi\in\Gamma} \varphi$$

A formula of the form $\langle a\rangle\varphi$ is equivalent to $(a) \rightarrow\{tt, \varphi\}$ and a formula of the form $[a]\varphi$ is equivalent to $(a) \rightarrow\emptyset \vee (a) \rightarrow\{\varphi\}$. This means that every μ-calculus formulas can be rewritten using the operator $(a) \rightarrow\Gamma$.

Definition 29 The set of disjunctive formulas, dF_μ is the smallest set defined by the following clauses:

- tt, ff, X belongs to dF_μ.
- If $\varphi, \psi \in dF_\mu$ then $\varphi \vee \psi \in dF_\mu$; if moreover X does not occur in a sub formula of φ of the form $X \wedge \gamma$, then $\mu X.\varphi(X)$, $\nu X.\varphi(X) \in dF_\mu$.
- Formula $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n \in dF_\mu$ provided that every φ_i is in $\{tt, ff\}$ or a formula of the form $\varphi_i = (a_i) \rightarrow\Theta_i$ with $\Theta_i \subseteq dF_\mu$. It is required that for any event a there can be at most one conjunct of the form $(a) \rightarrow\Gamma$ among $\varphi_1, \varphi_2, \ldots, \varphi_n$.

Theorem 30 ([JW95]) *For every formula φ, there exists an equivalent disjunctive formula ψ.*

1.4 Alternating Tree Automata

We present alternating tree automata [MS87] and non deterministic tree automata. Alternating tree automata recognize labelled transition systems. They are a main technical tool for proofs and, in understanding of the μ-calculus. Alternating tree automata have the same expressive power as the μ-calculus in the sense that every μ-calculus formula can be translated into an equivalent alternating tree automaton and reciprocally. The definition of alternating tree automata presented in this section is different from the one in [MS87]. In the definition presented below, we use modal operators $(\langle\rangle, [])$ of the μ-calculus in transition relations.

Definition 31 An alternating tree automaton is the structure $\mathcal{A} = \langle Q, \Sigma, q^0, \Delta, Acc\rangle$ where Q is a finite set of states, Σ is an alphabet, q^0 is the initial state, $\Delta : Q \rightarrow TF(Q, \Sigma)$ is a transition relation which assigns a transition formula to each state of the automaton, and Acc is the parity condition given by a function $\Omega_\mathcal{A} : Q \rightarrow \{0, \ldots, k\}$. The set $TF(Q, \Sigma)$ of *transition formulas* is defined as follows:

- tt and ff are transition formulas.
- For every $q \in Q$, $\langle a \rangle q$, $[a]q$ are transition formulas.
- for every q_1, $q_2 \in Q$, $q_1 \wedge q_2$ and $q_1 \vee q_2$ are transition formulas

An alternating tree automaton as above accepts labelled transition systems. The meaning of alternating tree automata can be defined using the game approach. To decide whether a transition system is accepted by an alternating tree automaton, one can consider a two player parity game. Let $\mathcal{S} = \langle S, \Sigma, s^0, \Delta_S \rangle$ be a Σ-labelled transition system. The acceptance game of $\mathcal{A}$ over $\mathcal{S}$ is the tuple $\mathcal{G}(\mathcal{A}, S) = \langle N, N_E, N_A, T, Acc_{\mathcal{G}} \rangle$ where:

- $N_E = S \times \mathcal{F}^E$ and $\mathcal{F}^E \subseteq TF(Q, \Sigma)$ is the set of transition formulas of the form ff, $\varphi \vee \psi$, $\langle a \rangle \varphi$ with $\varphi, \psi \in TF(Q, \Sigma)$.
- $N_A = S \times TF(Q, \Sigma) \setminus N_E$
- There is no move from (s, tt) nor from (s, ff), for every $s \in Q$.
- From $(s, \varphi \wedge \psi)$ as well as from $(s, \varphi \vee \psi)$ there are moves to (s, φ) and to (s, ψ).
- From $(s, [a]\varphi)$ and from $(s, \langle a \rangle \varphi)$ there are moves to (s', φ) for every s' such that $s \xrightarrow{a} s'$.
- There is a move from (s, q) to $(s, \Delta(q))$.
- $Acc_{\mathcal{G}}$ is the max-parity condition characterised by the function $\Omega_{\mathcal{G}} : N \to \mathbb{N}$ defined by

$$\Omega_{\mathcal{G}}(s, \varphi) = \begin{cases} 0 & \text{if } \varphi \text{ is not a state} \\ \Omega_{\mathcal{A}}(q) & \text{if } \varphi = q \text{ is a state} \end{cases}$$

We say that $\mathcal{A}$ accepts $\mathcal{S}$ (or $\mathcal{S}$ models $\mathcal{A}$), and we write $\mathcal{S} \in \mathcal{L}(\mathcal{A})$ or $\mathcal{S} \models \mathcal{A}$, if there is a winning strategy for the player *Eve* in $\mathcal{G}(\mathcal{S}, \mathcal{A})$.

Theorem 32 ([Niw88, EJ91, Wil01]) *For every μ-calculus formula φ there is an alternating tree automaton $\mathcal{A}_\varphi$ such that for every Σ-labelled transition system $\mathcal{S}$:*

$$\mathcal{S} \models \varphi \text{ if and only if } \mathcal{S} \in \mathcal{L}(\mathcal{A}_\varphi).$$

Conversely, for every alternating tree automaton $\mathcal{A}$ there is a μ-calculus formula $\varphi_{\mathcal{A}}$ such that every Σ-labelled transition system $\mathcal{P}$:

$$\mathcal{S} \in \mathcal{L}(\mathcal{A}) \text{ if and only if } \mathcal{S} \models \varphi_{\mathcal{A}}.$$

1.5 Frameworks for Discrete-Time Control

We present the Ramadge et al. approach and the Arnold et al. approach to the discrete-time control problem. We recall that in discrete-time control, timing information are not explicit in models of systems and specifications. In the Ramadge et al. Approach specifications are linear-time properties described with an automaton on words. The Arnold et al. approach considers branching-time properties described with μ-calculus formula.

1.5.1 The Ramadge et al. Approach

The controller synthesis problem considered by Ramadge and Wonham [RW89] consists to provide controller for a system (called the plant). The main goal is that the system resulting from the combination of the plant with controller, the *controlled system*, satisfies the given requirement. In [RW89], Ramadge and Wonham consider centralised and decentralised controller synthesis problems with full and partial observability hypothesis on events. We only present their framework for the centralised controller synthesis with total observation.

In that framework, plants and controllers are deterministic finite state automata over a set of events Σ partitioned into a set of controllable event Σ_c and a set of uncontrollable events Σ_u. Requirements controlled systems are given with regular languages (on words). Controllers never forbid an occurrence of uncontrollable event. The synthesis of a controller is effective and polynomial in the size of the plant and the requirement. .

Since the works in [RW89], the control problem has been studied in more powerful settings. Some of these studies are presented in the following.

1.5.2 The Arnold et al. Approach

The framework of Arnold et al. [AVW03, ABPV05, AW07] for the supervisory control consider transition system models and control objective described with alternating tree automata on transition systems, or equivalently μ-calculus formulas. There, the notions of controllability, observability and distinguishability are considered for centralised and decentralised controller synthesis problems. We will just present the framework for the centralised controller synthesis when the set of events Σ, occurring in the plants are partitioned into a set Σ_c of controllable events and a set Σ_u of uncontrollable events.

Plants and controllers are finite state labelled transition systems (not necessarily deterministic). A controller has to satisfy the untimed control condition (UCC) that requires that it can not forbid any occurrence of an uncontrollable event. This property for controllers can be described with an alternating tree automaton (see below).

The centralised controller synthesis problem is:

given a plant $\mathcal{P}$ and two μ-calculus formulas φ and ψ, does there exists a controller $\mathcal{R}$ satisfying the condition (UCC) such that $\mathcal{P} \times \mathcal{R}$ satisfies φ and $\mathcal{R}$ satisfies ψ?

The solution to the centralised controller synthesis uses a notion of quotient of a control objective φ with a plant $\mathcal{P}$. Because a μ-calculus formula can be translated into an equivalent alternating tree automaton on transition systems (see Theorem 32), Arnold et al. assume that the control objective φ is described with an equivalent alternating tree automaton $\mathcal{A}_\varphi$. Then they provide a quotient operator $\mathcal{A}_\varphi/\mathcal{P}$ of $\mathcal{A}_\varphi$ with a plant $\mathcal{P}$. The quotient operation is defined in such a way that it satisfies the property presented in Proposition 33 just below.

Proposition 33 ([AVW03, AW07]) Given an alternating tree automaton $\mathcal{A}$, two finite state transition systems $\mathcal{P}$ and $\mathcal{R}$, there is an alternating tree automaton $\mathcal{A}/\mathcal{P}$ such that:

$$\mathcal{R} \models \mathcal{A}/\mathcal{P} \text{ if and only if } \mathcal{P} \times \mathcal{R} \models \mathcal{A}$$

To ensure $\mathcal{R}$ in the proposition above to be a controller, $\mathcal{R}$ should additionally satisfies the untimed controller conditions (UCC). Arnold et al. [ABPV05] have shown that the control condition (UCC) can be described with a μ-calculus formula

$$\nu X. \bigwedge_{a \in \Sigma_u} \langle a \rangle X \wedge \bigwedge_{a \in \Sigma_c} [a] X$$

that is equivalent to the one state alternating tree automaton $\mathcal{B}$ defined as follows:

$$\mathcal{B} = \langle \{q^0\}, \Sigma, q^0, \Delta, Acc \rangle \text{ where}$$

$$\Delta(q^0) = \bigwedge_{a \in \Sigma_u} \langle a \rangle q^0 \wedge \bigwedge_{a \in \Sigma_c} [a] q^0.$$

where Acc is the parity condition given by a function $rank$ that assigns the value 0 to the state q^0.
Let us comment the transition formula of the one state modal automaton $\mathcal{B}$. The formula has two parts. The first part is the conjunction $\bigwedge_{a \in \Sigma_u} \langle a \rangle q^0$. A state in which this part is true should (because of the existential modality) have, for every uncontrollable event from Σ_u, an outgoing transition to a state in which $\Delta(q^0)$ is true again. The second part is the conjunction $\bigwedge_{a \in \Sigma_c} [a] q^0$; it requires every successor of a state (satisfying the formula), with a controllable event, to satisfy $\Delta(q^0)$; it does not requires its models to have transitions labelled with controllable events from Σ_c. The parity index of q^0 is even and it is equal to 0. In consequence every infinite path in the acceptance game is accepted.

As alternating tree automata are closed under intersection, a controller for a plant $\mathcal{P}$ under a specification $\mathcal{A}_\varphi$ should satisfy $\mathcal{A}_\varphi / \mathcal{P} \cap \mathcal{A}_\psi \cap \mathcal{B}$. This provides a hint of a proof for the decidability of the centralised controller synthesis problem.

Theorem 34 ([AVW03, AW07]) *Given a plant $\mathcal{P}$ and two μ-calculus formulas φ and ψ, the problem of checking whether there exists a controller $\mathcal{R}$ satisfying the condition (UCC) such that $\mathcal{P} \times \mathcal{R} \models \varphi$ and $\mathcal{R} \models \psi$ is decidable. Moreover, the computation of a witness controller is effective.*

1.6 Frameworks for Dense-Time Supervisory Control

We present the approach of Madhusudan et al. [DM02] and the approach of Laroussinie et al. [BCL05] to the controller synthesis of dense-time systems. These approaches consider timed automata model [AD94] that we present in the next subsection.

1.6.1 The Timed Automata Model

Let $\mathcal{H}$ be a set of clocks. A clock constraint is a comparison of a clock, or the difference between two clocks, with a constant. Let $Gds_{\mathcal{H}}$ be a set of clock constraints. Clocks are real-valued variables. If v represents a valuation, $v(h)$ represents the value of the clock h, $(v+t)(h)$ gives the value of the clock h after a delay of t time units, and $v[H := 0]$ resets every clock in H.

Definition 35 A timed automaton over $(\mathcal{H}, \Sigma)$ is a structure $\mathcal{P} = \langle P, \mathcal{H}, \Sigma, p^0, \Delta_P, Acc\rangle$ where P is a finite set of states, Σ is an alphabet, p^0 is the initial state, $\Delta_P \subseteq P \times Gds_{\mathcal{H}} \times \Sigma \times 2^{\mathcal{H}} \times P$ is the transition relation and Acc is the acceptance condition.
A timed automaton is deterministic if there are no two distinct transitions of the form $p \xrightarrow{g',a,H'} p'$ and $p \xrightarrow{g'',a,H''} p''$ such that g'' and g' can be satisfied by the same valuation of the clocks.

A timed automaton $\mathcal{P}$ as defined above represents a transition system whose states are pairs of the form (p, v) made of a state of the timed automaton and a valuation. Transitions in the transition system are of the form $(p, v) \xrightarrow{t} (p, v + t)$ or $(p, v) \xrightarrow{a} (p', v[H := 0])$. A transition $(p, v) \xrightarrow{t} (p, v + t)$ represents a delay (of amount $t \in \mathbb{R}^+$) that occurs in the timed transition system when it is in state p and the values of the clocks are given by v. A transition $(p, v) \xrightarrow{a} (p, v[H := 0])$ indicates that the system moves from the state p to the state p' when the event a occurs; and then it immediately resets all the clocks in H. The latter transition is possible if the timed transition system has a transition $p \xrightarrow{g,a,H} p'$ and the values of the clocks, given by v, satisfy the constraint g.

A timed automaton accepts timed words. A *timed word* over an alphabet Σ is sequence $w = (a_i, t_i)_{i=1..}$ such that $i < j$ implies $t_i \leq t_j$.

A *run* ρ, of a timed automaton over a timed word $(a_i, t_i)_{i=1..}$ is a sequence of the form

$$\rho = (p_0, v_0) \xrightarrow{a_0,t_0} (p_1, v_1) \xrightarrow{a_1,t_1} \dots \xrightarrow{a_i,t_i} (p_{i+1}, v_{i+1}) \dots$$

with $p_i \in P$, v_i is a valuation of the clocks, for all $i \geq 0$, satisfying the following requirements:

- $p_0 = p^0$ is the initial state of the automaton.
- $v_0(h) = 0$ for all $h \in \mathcal{H}$.
- for all $i \geq 0$, there is a transition $p_i \xrightarrow{g_i,a_i,H_i} p_{i+1}$ such that $v_i + t_i - t_{i-1}$ satisfies g_i and v_{i+1} equals $v_i + t_i - t_{i-1}[H_i := 0]$.

A *run is accepting* if and only if its projection on the states (P) of the timed automaton belongs the acceptance condition (Acc) of the automaton. A timed word, w is accepted by a timed automaton if and only if there is an accepting run of the automaton over w.

The *language* of a timed automaton, $\mathcal{L}(\mathcal{A})$ is the set of timed words over which there is an accepting run. Formally,

$$\mathcal{L}(\mathcal{A}) = \{w \,|\, w \text{ is a timed word and } \mathcal{A} \text{ accepts } w\}$$

The following theorem presents some fundamental results on languages of timed automata. Automata in that theorem are timed automata with the Büchi acceptance condition. These results are useful for understanding the results presented in the next subsection.

Theorem 36 ([AD94]) *Emptiness testing is decidable for non deterministic timed automata. Timed automata are closed under union and intersection. Non deterministic timed automata are not closed under complementation but deterministic timed automata are closed under complementation. The inclusion testing between non deterministic timed automata is undecidable but, it is decidable to check whether a timed automata is included in a deterministic timed automata.*

1.6.2 The Madhusudan et al. Approach for Automata Specification

Madhusudan et al. [DM02] consider the controller synthesis for timed specifications. A plant $\mathcal{P}$ is a deterministic timed automaton over $(\mathcal{H}_\mathcal{P}, \Sigma)$ where, $\mathcal{H}_\mathcal{P}$ denotes the set of clocks used by the plant and the set of events $\Sigma = \Sigma_u \cup \Sigma_c$ is partitioned into a set Σ_c of controllable events and a set Σ_u of uncontrollable events. A controller $\mathcal{S}$ is a deterministic timed automaton over $(\mathcal{H}_\mathcal{P} \cup \mathcal{H}_\mathcal{S}, \Sigma)$ where, $\mathcal{H}_\mathcal{S}$ is a set of clocks disjoint from $\mathcal{H}_\mathcal{P}$. A controller is combined with the plant for satisfying a control objective.

The notion of product between timed automata formalises the combination between systems (the plant and the controller). This notion is defined as follows:

Definition 37 The *product* of a timed automaton $\mathcal{P} = \langle P, H_1, \Sigma, p^0, \Delta_P, Acc_1 \rangle$ with a timed automaton $\mathcal{S} = \langle S, H_2, \Sigma, s^0, \Delta_\mathcal{S}, Acc_2 \rangle$ is the timed automaton $\mathcal{P} \times \mathcal{S} = \langle P \times S, H_1 \cup H_2, \Sigma, (p^0, s^0), \Delta, Acc \rangle$ where, and Δ is given by $(p, s) \overset{g,a,X\cup Y}{\longrightarrow} (p', s') \in \Delta$ if and only if $p \overset{g_1,a,X}{\longrightarrow} p'$, and $s \overset{g_2,a,Y}{\longrightarrow} s'$ with g representing the conjunction of g_1 and g_2.

A controller satisfies the following timed control conditions (TCC):

(TCC) (C1) $\mathcal{S}$ has resets only in $\mathcal{H}_\mathcal{S}$ (i.e, if $s \overset{g,a,H}{\longrightarrow} s'$, then $H \subseteq \mathcal{H}_\mathcal{S}$).

(C2) $\mathcal{S}$ does not restrict uncontrollable events (non restricting): whenever we have $w \in \mathcal{L}(\mathcal{P} \times \mathcal{S})$ and $(w.(a,t)) \in \mathcal{L}(\mathcal{P})$ with $a \in \Sigma_u$, then $w.(a,t) \in \mathcal{L}(\mathcal{P} \times \mathcal{S})$.

(C3) $\mathcal{S}$ is *non-blocking*: whenever we have $w \in \mathcal{L}(\mathcal{P} \times \mathcal{S})$ and $(w.(b,t)) \in \mathcal{L}(\mathcal{P})$, there exists $c \in \Sigma$ and $t' \in \mathbb{R}^+$ such that $(w.(c,t')) \in \mathcal{L}(\mathcal{P} \times \mathcal{S})$.

A control objective is described by a timed automaton with Büchi acceptance condition. It can describe a set of undesired behaviours or a set of desired behaviours. The timed automata can be deterministic or not.

The controller synthesis against undesired behaviours is: given a plant $\mathcal{P}$ and a timed automaton $\mathcal{A}$, does there exists a controller $\mathcal{S}$ for $\mathcal{P}$ such that $\mathcal{L}(\mathcal{P} \times \mathcal{S}) \cap \mathcal{L}(\mathcal{A}) = \emptyset$? The control synthesis against desired behaviours is: given a plant $\mathcal{P}$ and a timed automaton $\mathcal{A}$, does there exists a controller $\mathcal{S}$ for $\mathcal{P}$ such that $\mathcal{L}(\mathcal{P} \times \mathcal{S}) \subseteq \mathcal{L}(\mathcal{A})$?

In Table 1, we present decidability results [DM02] for the controller synthesis problem against undesired behaviours and the controller synthesis problem for desired behaviours; for decidable cases, a finite-state controller can be synthesized.

Limited resources			Unlimited resources		
Det. Cont. Obj	Nondet. Cont. Obj.		Det. Cont. Obj	Nondet. Cont. Obj.	
	Desired	Undesired		Desired	Undesired
Decidable	Undecidable	Decidable	Decidable	Undecidable	Undecidable

Table 1: Controller Synthesis results for Madhusudan et al..

Theses results do not only depend on whether the specification is deterministic or not; but they also depend on some hypothesis made on the number of clocks and the constants that

appear in the controller; these latter parameters are called the resources of the controllers. In the case of limited resources, a maximal constant in the controller is specified and the set $\mathcal{H}_{\mathcal{S}}$ is also specified.

1.6.3 The Laroussinie et al. Approach for L_ν Specification

Laroussinie et al. [BCL05] use the framework of timed automata to describe plants and the timed branching-time logic L_ν to describe internal control objectives.

Syntax and Semantics of L_ν

Definition 38 The logic L_ν over the finite set of clocks $\mathcal{H}$, the set of identifiers Var, and the set of events Σ is defined as the set of formulas generated by the following grammar:

$$\varphi ::= tt \,|\, ff \,|\, \varphi \vee \varphi \,|\, \varphi \wedge \varphi \,|\, h \underline{in} \,\varphi \,|\, h \bowtie c \,|\, \langle a \rangle \varphi \,|\, [a]\varphi \,|\, \langle \delta \rangle \varphi \,|\, [\delta]\varphi \,|\, X$$

where, $a \in \Sigma$ is an event, $h \in \mathcal{H}$ is a clock variable, $c \in \mathbb{Q}_{\geq 0}$ is a constant, X is a variable, $\bowtie \in \{\leq, \geq, <, >\}$.

The logic L_ν allows for the recursive definition of formulas by including a set Var of variables. The formula associated with each of the identifiers is specified by a declaration $\mathcal{D}$; In other words, the declaration $\mathcal{D}$ assigns a L_ν formula to each identifier. For an identifier X, we write $X \stackrel{def}{=} \varphi$ if $\mathcal{D}(X) = \varphi$. Intuitively X stands for the largest solution of the equation $X \stackrel{def}{=} \varphi$.

A formula is interpreted over the semantics of timed automaton. From what has preceded, we use the notion $\mathcal{P} \vDash_{\mathcal{D}} \varphi$ to say that the timed automaton $\mathcal{P}$ is a model of φ with respect to the declaration $\mathcal{D}$. Let us take a timed automaton $\mathcal{P}$, whose set of clocks $\mathcal{K}$ is disjoint from the set of clocks $\mathcal{H}$ occurring in formulas. Formulas are interpreted over *extended states* of the form (p, v) where, p is a state of $\mathcal{P}$, v is a valuation of all clocks in $\mathcal{K} \cup \mathcal{H}$. The satisfaction relation $\vDash_{\mathcal{D}}$ is the largest relation satisfying the following implications:

- it is true that $\mathcal{P}, (p, v) \vDash_{\mathcal{D}} tt$.
- it is false that $\mathcal{P}, (p, v) \vDash_{\mathcal{D}} ff$.
- if $\mathcal{P}, (p, v) \vDash_{\mathcal{D}} \varphi \vee \psi$ then $\mathcal{P}, (p, v) \vDash_{\mathcal{D}} \varphi$ or $\mathcal{P}, (p, v) \vDash_{\mathcal{D}} \psi$
- if $\mathcal{P}, (p, v) \vDash_{\mathcal{D}} \varphi \wedge \psi$ then $\mathcal{P}, (p, v) \vDash_{\mathcal{D}} \varphi$ and $\mathcal{P}, (p, v) \vDash_{\mathcal{D}} \psi$
- if $\mathcal{P}, (p, v) \vDash_{\mathcal{D}} h \bowtie c$ then $v(h) \bowtie c$.
- if $\mathcal{P}, (p, v) \vDash_{\mathcal{D}} [a]\varphi$ then for all $p \stackrel{g,a,H}{\longrightarrow} p'$ such that v satisfies g we have $\mathcal{P}, (p', v[H := 0]) \vDash_{\mathcal{D}} \varphi$.
- if $\mathcal{P}, (p, v) \vDash_{\mathcal{D}} \langle a \rangle \varphi$ then there is $p \stackrel{g,a,H}{\longrightarrow} p'$ such that v satisfies g and $\mathcal{P}, (p', v[H := 0]) \vDash_{\mathcal{D}} \varphi$.

- if $\mathcal{P}, (p, v) \models_{\mathcal{D}} [\delta]\varphi$ then for all $t \in \mathbb{R}^+$ $\mathcal{P}, (p, v + t) \models_{\mathcal{D}} \varphi$.
- if $\mathcal{P}, (p, v) \models_{\mathcal{D}} \langle\delta\rangle\varphi$ then there is $t \in \mathbb{R}^+$ such that $\mathcal{P}, (p, v + t) \models_{\mathcal{D}} \varphi$.
- if $\mathcal{P}, (p, v) \models_{\mathcal{D}} h \underline{in} \varphi$ then $\mathcal{P}, (p, v[\{h\} := 0]) \models_{\mathcal{D}} \varphi$
- if $\mathcal{P}, (p, v) \models_{\mathcal{D}} X$ then $\mathcal{P}, (p, v) \models_{\mathcal{D}} \mathcal{D}(X)$

Any relation satisfying the above implications is referred to as a satisfiability relation. The relation $\models_{\mathcal{D}}$ is the union of all satisfiability relations.

The control problem

Events are controllable or non controllable. A plant $\mathcal{P}$ is a timed automaton that is deterministic with respect to controllable events. At any time and in any state, the time elapses or an event occurs. The plant does not control the occurrences of uncontrollable events.

The control objective is a formula φ of L_ν^{det} (L_ν^{det} is a deterministic fragment of L_ν). The fragment L_ν^{det} ensures that the conjunctions of L_ν^{det} formulas are in some sense deterministic and thus, they can be merged safely. By this way, a controller against a control objective $\varphi \wedge \psi$ will "easily" combine a controller against φ and a controller against ψ.

A controller for a plant is a function f that during the execution of the system constantly gives information about what should be done in order to ensure the control objective. A controller can not prevent uncontrollable events from occurring; but it can disable a controllable event at any time. We write $f(\mathcal{P})$ for the controlled system.

The control problem considered by Laroussinie et al. is the following.

> Given a timed automaton $\mathcal{P}$, the plant, and a L_ν^{det} formula φ, a (deterministic) control objective, is there a controller f such that $f(\mathcal{P}) \models \varphi$?

The main result of Laroussinie et al. is that the control problem can be reduced to the standard model-checking problem. For that purpose, from a L_ν^{det} control objective φ, the formula $\overline{\varphi} = \bigvee_{e \in \Sigma_c \cup \{\lambda\}} \overline{\varphi}^e$ is defined. For a controllable event a_c the formula $\overline{\varphi}^{a_c}$ will hold when there is a controller which ensures φ and which starts by enforcing the event a_c. The formula $\overline{\varphi}^\lambda$ that will hold when there is a controller which ensures φ and which starts by delaying. The construction of these new formulas involves the introduction a new modal operator $[\delta\rangle$ that can not be described using a L_ν formula and whose semantics is the following.

> $\mathcal{P}, (p, v) \models_{\mathcal{D}} \varphi[\delta\rangle\psi$ if and only if either $\forall t \in \mathbb{R}^+$, $\mathcal{P}, (p, v + t) \models_{\mathcal{D}} \varphi$ or, $\exists t \in \mathbb{R}^+$ such that $\mathcal{P}, (p, v + t) \models_{\mathcal{D}} \psi$ and $\forall 0 \leq t' < t$, $\mathcal{P}, (p, v + t) \models_{\mathcal{D}} \varphi$.

The resulting logic (L_ν augmented with $[\delta\rangle$), L_ν^{cont} enables to express dense-time control requirement: some property is true for a subset of the states of the plant that are reachable by time elapsing before a controllable action leading to good states is possible. Thus, checking the existence of a controller for a timed automaton against a L_ν^{det} control objective is reduced to checking whether the timed automaton satisfies a dense-time control requirement which is itself described with a L_ν^{cont} formula.

Theorem 39 ([BCL05]) *For all $\varphi \in L_\nu^{det}$ and a timed automaton $\mathcal{P}$, there exists a controller f such that $f(\mathcal{P}) \models_{\mathcal{D}} \varphi$ if and only if $\mathcal{P} \models_{\mathcal{D}} \overline{\varphi}$.*

The decidability of the control problem (the construction of a witness controller is not effective) comes from the following theorem.

Theorem 40 ([BCL05]) *Given $\psi \in L_\nu^{cont}$, and a timed automaton $\mathcal{P}$, it is EXPTIME-complete to decide whether $\mathcal{P}$ is a model of ψ.*

Chapter 2

Timed Processes

In reactive real-time systems, the correctness of the tasks they perform depends not only upon logical correctness, but also upon the times at which the tasks are performed. For recording the duration of a task, we can use an event that is triggered at the beginning of the task and another event that is triggered when the task ends; then the duration of the task will be the difference between the time at which the termination event happens and the time at which the beginning event happens. If there are several tasks in the system, we could imagine that there are as many start-events as tasks. The dynamics of a real-time system can be described through the variation of its tasks. A variation can be constrained with the occurrence of events and/or timing information.

In this chapter, we consider models for a class of real-time systems and models for representing their behaviours. We called our models *timed processes*. Timed processes have local clocks each associated to an event and such a clock gathers the time elapsed since the last occurrence of the corresponding event. A timed process is a finite state labelled transition system whose transitions are labelled with constraints on clocks and events. A constraint on clocks is just a conjunction of comparisons of values of a clock or the difference between two clock values, with an integer constant. The latter is are called *diagonal constraint*.

Clock are interpreted over real numbers. The value of each clock grows continuously and with the same rate as the time unless it is reset. A timed process is a finite representation of all the behaviours of a real-time system. A behaviour of a real-time system is a succession of states of its timed process paired with the values of clocks. When the system is in some state, the time elapses continuously (the values of the clocks too) until an event occurs. Then, the process instantaneously selects a transition labelled with that event and checks whether the constraint on the chosen transition is satisfied by the values of clocks before it resets the clock associated to the event and moves to the target state of the transition. If the constraint is not satisfied, the process does not change the state.

We will use transition systems to represent the semantics (set of behaviours) of timed processes. A transition in the semantics of a timed process will be labelled with an event or a valuation of clocks; a state will be a pair made of a state of the timed process and the values of the clocks. As clocks values are real-numbers, semantics are infinite state labelled transition systems, and each state has infinitely many outgoing transitions. Because infinite

models are difficult to handle, we introduce two representations for semantics called *the M-action representation* and the *M-region representation*.

The M-action representation will be obtained from the semantics by replacing each transition labelled with a valuation by a transition labelled with canonical and atomic constraints satisfied in that valuation. We will require constraints to use constants lower than M. As there will be finitely many canonical and atomic constraints, each state will have finitely many outgoing transitions, but there will still be infinitely many states in M-action representations. Then, we introduce M-region representation a state of which is a state of the underlying timed process paired with a M-region. A region is just a set of "equivalent" valuations. There are finitely many regions and then, M-region representations are finite structures suitable suitable for verification purposes. As we will show, using bisimulation relation, M-region representations preserve behavioral properties of the semantics.

We will consider that systems communicate in a synchronous mode. For communicating systems in synchronous mode, an event must happen at the same time in all the systems in order to be considered. We will assume that the communicating devices will be 0-delay. We will formalise the communication by defining a product operation between timed processes. We will show that the semantics of the product of two timed processes is "the same" as the product of the semantics of that processes.

A natural and fundamental problem that arises when defining models for systems is the reachability problem. The reachability problem requires to check if a target state could be reached from a source state when a system executes. There are two approaches to this problem: the forward analysis, and the backward analysis. We will present an algorithm based on the M-region representation that is correct for the backward and the forward approaches whatever is the nature or the constraint (general or diagonal free). But considering the zone-based representation of the timing context (a zone is just a set of valuations satisfying a constraint), we show the incorrectness of the forward analysis algorithm when diagonal constraints are authorised in the timed processes.

Related Results: All results presented in this section are known. Timed processes are nothing else but event-recording automata [AFH99] without an acceptance condition these in turn are a subclass of timed automata [AD94]. The reachability problem for timed automata has been considered using region abstraction [ACD+92, LY97] and zone abstraction [LPY97, BY04] and algorithms for the reachability problem have been implemented in verification tools like Uppaal [LPY97, BLL+96, BDL04] or Kronos [BTY97, Yov98]. Bouyer has shown [Bou03] (see also [BLR05]) the incorrectness of a zone-approach for the reachability problem of timed automata with diagonal constraints.

This chapter is organised as follows. In the next section we consider clocks, constraints, we also present decomposition of constraints into atomic constraints. In Section 2.2 we present regions and their properties. Zones and their operations are presented in Section 2.3. In Section 2.4, we define timed processes, their semantics and representations of semantics. We use some properties of regions to show that M-region representation can be used instead of the semantics. We present the the product of timed processes in Section 2.5, and in Section 2.6 we consider the reachability analysis.

2.1 Clock, Valuation, Constraints

We define clocks that are real numbers valued variables. We also define clock constraints and we present their decomposition into atomic clock constraints.

2.1.1 Clocks and Valuations

Clocks are variables evaluated over real numbers. There are two operations on time, the time elapse operation that gives the value of the clock after a delay and the reset operation that sets the value of a clocks to 0.

Let $\mathbb{R}^+$ be the set of non negative real numbers. We consider $\mathcal{H} = \{h_1, h_2, \dots\}$ a set of clocks variables (or clocks for simplicity).

Definition 41 A *valuation* on a set of clock $\mathcal{H}$ is a total function $v : \mathcal{H} \to \mathbb{R}^+$.

The symbol $\mathcal{V}$ represents the set of valuations. Given a valuation $v \in \mathcal{V}$, and a clock $h \in \mathcal{H}$, the valuation $v+t$ is defined by $[v+t](h) = v(h)+t$ and, the valuation $v[h := 0]$ is defined by $v[h := 0](h') = 0$ if $h = h'$ else $v[h := 0](h') = v(h')$. We say that a valuation v is a *successor* of a valuation v' if $v = v' + t$ for some $t \in \mathbb{R}^+$.

Example: Let $\mathcal{H} = \{h_1, h_2\}$ be a set of two clocks. In Table 2, we present some valuations on h are some valuation on $\mathcal{H}$.

$$\begin{cases} v_0(h_1) = 0 \\ v_0(h_2) = 0 \end{cases} \quad \begin{cases} v_1(h_1) = 0.35 \\ v_1(h_2) = 0.35 \end{cases} \quad \begin{cases} v_2(h_1) = 0.35 \\ v_2(h_2) = 0 \end{cases} \quad \begin{cases} v_3(h_1) = 0.85 \\ v_3(h_2) = 0.50 \end{cases}$$

$$\begin{cases} v_4(h_1) = 0 \\ v_4(h_2) = 0.50 \end{cases} \quad \begin{cases} v_5(h_1) = 0.35 \\ v_5(h_2) = 0.85 \end{cases}$$

Table 2: Examples of valuations.

These valuations are such that $v_1 = v_0 + 0.35$, $v_2 = v_1[h_2 := 0]$, $v_3 = v_2 + 0.50$, $v_4 = v_3[h_1 := 0]$, $v_5 = v_4 + 0.35$ and $v_2 = v_5[h_2 := 0]$. In Figure 2 we give another representations of these valuations in Cartesian reference.

□

2.1.2 Constraints

Constraints are conjunctions of simple constraints; and a simple constraint is a comparison of a clock with an integer (diagonal free simple constraint) or a comparison of the difference between two clocks with and integer. Diagonal free constraints use only diagonal free simple constraints. Constraints are interpreted over valuations. The semantics of a constraint is the set of valuations satisfying it. We will also consider two types of *atomic constraints* : *rectangular constraints* and triangular constraints.

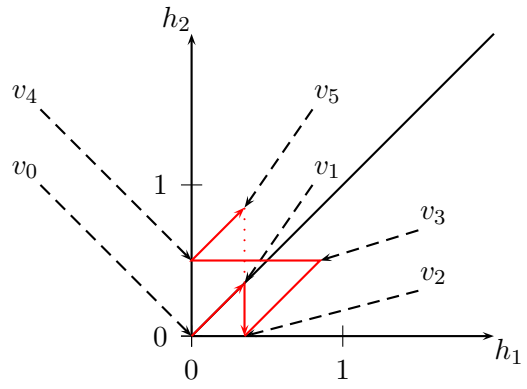

Figure 2: Representation of valuations in Cartesian reference.

Definition 42 A *simple constraint* defined on a set of clocks $\mathcal{H}$ is an equation of the form $h - h' \bowtie n$ or $h \bowtie n$ where $n \in \mathbb{N}$, $\bowtie$ is one of $\{<, \leq, \geq, >\}$ and $h, h' \in \mathcal{H}$.
A *diagonal free simple constraint* is a simple constraint of the form $h \bowtie n$.

Definition 43 A *clock constraint* over a set of clocks $\mathcal{H}$ is a conjunction of simple constraints. $\Phi_{\mathcal{H}}$, denotes the set of clock constraints over $\mathcal{H}$. A *diagonal-free clock constraint* is a clock constraint that uses only diagonal free simple constraints. $Gds_{\mathcal{H}}$ denotes the set of diagonal-free clock constraints over $\mathcal{H}$.

We will often write $h = n$ or $h - h' = n$ as an abbreviation of $h \leq n \wedge h \geq n$. We also write $h - h' = n$ to represent the constraint $h - h' \leq n \wedge h - h' \geq n$.

Later we consider two special clock constraints tt and ff defined by: $tt = \bigwedge_{h \in \mathcal{H}} h \geq 0$ and $ff = \bigwedge_{h \in \mathcal{H}} h < 0$.

The notion of a constraint satisfied in a given valuation denoted $v \vDash g$ is defined inductively as follows:

- $v \vDash h \bowtie n$ if and only if $v(h) \bowtie n$
- $v \vDash h - h' \bowtie n$ if and only if $v(h) - v(h') \bowtie n$
- $v \vDash g_1 \wedge g_2$ if and only if $v \vDash g_1$ and $v \vDash g_2$

The meaning of a constraint g, denoted $[\![g]\!]$, is the set of valuations in which it is satisfied. Clearly, $[\![g]\!] = \{v : v \vDash g\}$. It becomes obvious that $[\![tt]\!] = \mathcal{H} \to \mathbb{R}^+$ and $[\![ff]\!] = \emptyset$.

Definition 44 A constraint g is inconsistent if $[\![g]\!] = \emptyset$.

Definition 45 The *bound* of a constraint g, denoted by M_g, is the maximal constant that appears in it. The bound of a set of constraints is the maximal value among the bounds of constraint it contains. A set of constraints is M-bounded if every constant in it is smaller than M.

Now we consider atomic constraints and we show how to decompose a constraint into an "equivalent" set of atomic constraints.

Definition 46 For a integer $M \in \mathbb{N}$, a *M-rectangular constraint* is a conjunction of the form $\bigwedge_{h\in\mathcal{H}} g_h$ where g_h is a constraint of the form $c < h < c+1$ or $h = c$ or $h > M$ with $c \in \mathbb{N} \cap [0..M[$.

The set of all M-rectangular constraints is denoted by $Agds_{\mathcal{H}}(M)$. The symbol $Agds_{\mathcal{H}}$ will denote the set $\bigcup_{M\in\mathbb{N}} Agds_{\mathcal{H}}(M)$

Definition 47 A *M-triangular constraint* is a conjunction of the form $\bigwedge_{h\in\mathcal{H}} g_h \wedge \bigwedge_{(h,h')\in\mathcal{H}^2} g_{h,h'}$ where $g_{h,h'}$ is a constraint of the forms $c < h - h' < c+1$ or $h - h' = c$ or $h - h' > M$ and g_h is of the form $c < h < c+1$ or $h = c$ or $h > M$ with $c \in \mathbb{N} \cap [0..M[$.

The symbol $Tgds_{\mathcal{H}}(M)$ denotes the set of all of M-triangular constraints. The symbol $Tgds_{\mathcal{H}}$ denotes the set $\bigcup_{M\in\mathbb{N}} Tgds_{\mathcal{H}}(M)$.

Notation: We often use the symbol $\hat{g}$ to denote a constraint in $Agds_{\mathcal{H}}(M)$ or $Tgds_{\mathcal{H}}(M)$ for some M. Later the terms *atomic constraints* will often be used in place of rectangular constraints or triangular constraints.

Let us first recall the following fact resulting from definitions of atomic constraints.

Fact 48 (atomicity) Let $M \in \mathbb{N}$ be a constant.

- $\forall \hat{g}, \hat{g}' \in Tgds_{\mathcal{H}}(M)$, if $[\![\hat{g}]\!] \neq [\![\hat{g}']\!]$ then $[\![\hat{g}]\!] \cap [\![\hat{g}']\!] = \emptyset$
- $\forall \hat{g}, \hat{g}' \in Agds_{\mathcal{H}}(M)$, if $[\![\hat{g}]\!] \neq [\![\hat{g}']\!]$ then $[\![\hat{g}]\!] \cap [\![\hat{g}']\!] = \emptyset$
- $\forall (\hat{g}, \hat{g}') \in Agds_{\mathcal{H}}(M) \times Tgds_{\mathcal{H}}(M)$, either $[\![\hat{g}']\!] \cap [\![\hat{g}]\!] = \emptyset$ or $[\![\hat{g}']\!] \subseteq [\![\hat{g}]\!]$

The first two items state that either the semantics of two atomic constraints of the same nature are equal, or they are disjoint. The last item of the above fact states that the semantics of a triangular constraint is either included in the semantics of a rectangular constraints, or the two semantics are disjoint.

Example: In Figure 3,we illustrate the concepts of constraints and diagonal free constraints. The constraints g_1 and g_3 are general constraints while the constraint g_2 is diagonal free. Moreover $[\![g_3]\!] = [\![g_1]\!] \wedge [\![g_2]\!]$. The constraint g_2 is a rectangular constraint in $Agds_{\mathcal{H}}(2)$ and the constraint g_3 is a triangular constraint. □

Normalization and Rectangularisation Until the end of this subsection we consider the decomposition of diagonal free constraint into set of rectangular constraints. We will need to consider constraints that do not involve constants greater than a fixed bound. For that purpose, we present the normalisation operation $norm_N$ that we use later to decompose constraints.

Definition 49 Given $N \in \mathbb{N}$, the *N-normalization* of a simple constraint C is the constraint $norm_N(C)$ defined by :

- $norm_N(h \bowtie n) = tt$ if $\bowtie \in \{<, \leq\}$ and $n > N$.

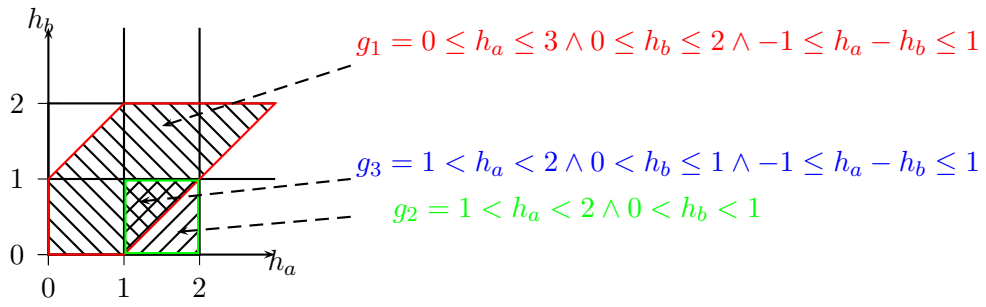

Figure 3: Illustration of constraints and diagonal free constraints.

- $norm_N(h - h' \bowtie n) = tt$ if $\bowtie \in \{<, \leq\}$ and $n > N$.
- $norm_N(h \bowtie n) = h > N$ if $\bowtie \in \{>, \geq\}$ and $n > N$.
- $norm_N(h - h' \bowtie n) = h - h' > N$ if $\bowtie \in \{>, \geq\}$ and $n > N$.
- In the other cases $norm_N$ does not modify the constraint.

Given a constraint g and an integer N, the *N-normalization* of g, $norm_N(g)$ is obtained by normalizing each simple constraint occurring in g.

Lemma 50 Let C, a diagonal-free simple constraint, there is a constant M such that:

- for every $N \geq M$, $[\![norm_M(C)]\!] = [\![norm_N(C)]\!] = [\![C]\!]$
- for every $N < M$, $[\![norm_M(C)]\!] \subsetneq [\![norm_N(C)]\!]$

Proof

1. When C has the form $h \bowtie n$ with $\bowtie \in \{<, \leq\}$ and consider $M = n$,
   (a) Let $N \geq M$, $norm_N(h \bowtie n)$ is equal to $norm_M(h \bowtie n)$ and they are equal to $h \bowtie n$ and we get the result that $[\![norm_M(C)]\!] = [\![norm_N(C)]\!] = [\![C]\!]$.
   (b) Let $N < M$, $norm_N(h \bowtie n) = h \geq 0$. Clearly $[\![norm_M(C)]\!] \subsetneq [\![norm_N(C)]\!]$.
2. When C has the form $h \bowtie n$ with $\bowtie \in \{>, \geq\}$ and consider $M = n$,
   (a) Let $N \geq M$, $norm_N(h \bowtie n)$ is equal to $norm_M(h \bowtie n)$ and they are equal to $h \bowtie n$ and we get the result that $[\![norm_M(C)]\!] = [\![norm_N(C)]\!] = [\![C]\!]$.
   (b) Let $N < M$, then $norm_N(h \bowtie n) = h \bowtie N$ and $[\![norm_M(C)]\!] = h \bowtie M$. Clearly, $[\![norm_M(C)]\!] \subsetneq [\![norm_N(C)]\!]$.

□

Let us recall that for a constraint g, M_g denotes the maximal constant occurring in g. We use the lemma above to show that the M-normalisation of a constraint does modify its semantics when M is greater or equal to M_g.

Proposition 51 Let $g \in Gds_{\mathcal{H}}$,

- for every $M \geq M_g$, $[\![norm_M(g)]\!] = [\![norm_N(g)]\!] = [\![g]\!]$
- for every $M < M_g$, $[\![norm_M(g)]\!] \subsetneq [\![norm_N(g)]\!]$

Proof
By definitions $g = \bigwedge_{i=1..n} C_i$ and, $[\![norm_M(g)]\!] = \bigcap_{i=1..n} [\![Norm_M(C_i)]\!]$. As M_g is greater that the constant used in every C_i, we get, using 50 that for $M \geq M_g$, $[\![norm_M(g)]\!] = [\![norm_N(g)]\!] = [\![g]\!]$and for $M < M_g$, $[\![norm_M(g)]\!] \subsetneq [\![norm_N(g)]\!]$ □

Example: Considering the constraint $g = 0 \leq h_a \leq 3 \wedge 0 \leq h_b \leq 2$, we present in Table 3 the results of M-normalisation operations depending on the value of M. It is easy to see that

M	$norm_M(g)$
0	tt
1	tt
2	$0 \leq h_b \leq 2$
3	$0 \leq h_a \leq 3 \wedge 0 \leq h_b \leq 2$

Table 3: Illustration of the normalisation operation.

for every $M < 2$, $[\![g]\!] \subseteq [\![norm_M(g)]\!]$ and for every $M \geq 2$, $[\![g]\!] = [\![norm_M(g)]\!]$ □

To obtain the decomposition of diagonal constraints, we firstly decompose diagonal free constraints into a set (possibly infinite) of unbounded rectangular constraints. Then, we use the normalisation procedure above on each atomic constraint in that set to have a finite set of bounded rectangular constraints. The decomposition of diagonal free constraints into a set of unbounded rectangular constraints is performed in two steps: in Lemma 52 we decompose simple diagonal free constraints and we use that decomposition in Proposition 53 to decompose diagonal free constraints.

Lemma 52 For every diagonal free simple constraint C, there is a set $Rect(C)$ of atomic diagonal free simple constraints such that $[\![C]\!] = \bigcup_{C' \in Rect(C)} [\![C']\!]$.

Proof
Let C be a diagonal free constraint C. We construct a set $Rect(C)$ depending on the form of C; and we show that for every $v \in \mathcal{V}$, $v \vDash C$ if and only if there is $C' \in Rect(C)$ such that $v \vDash C'$.

1. if C is of the form $h < n$ then set $Rect(C) = \{i < h < i+1, h = i \,|\, i = 0..n-1\}$
2. if C is of the form $h \leq n$ then set $Rect(C) = \{i < h < i+1, h = i \,|\, i = 0..n-1\} \cup \{h = n\}$
3. if C is of the form $h > n$ then set $Rect(C) = \{i < h < i+1, h = i+1 \,|\, i = n..\infty\}$

4. if C is of the form $h \geq n$ then set $Rect(C) = \{i < h < i{+}1, h = i{+}1 \mid i = n..\infty\} \cup \{h = n\}$

The proof that in each case, $[\![C]\!] = \cup_{C' \in Rect(C)} [\![C']\!]$, is obvious. □

We observe that simple constraints of the form $h > n$ to $h \geq n$ are decomposed into infinite set of constraints.

Proposition 53 For every diagonal-free constraint g, there is a set $Rect(g)$ of rectangular constraints such that $[\![g]\!] = \bigcup_{\hat{g} \in Rect(g)} [\![\hat{g}]\!]$.

Proof
The result is a consequence of the Lemma 52 above as a constraints is a conjunction of simple constraints. □

We say that $Rect(g)$ is the unbounded rectangular decomposition of g.

Now that we have decomposed diagonal free constraints into sets (possibly infinite) of unbounded rectangular constraints, we will apply the normalisation operation on each rectangular constraint in these sets; the result of the application of the normalisation operation with respect to a constant M will be finite set of M-rectangular constraints. But we need to show that the semantics of the constraint resulting from the application of the M-normalisation operation on a simple diagonal free constraint is the same as the union of the semantics of rectangular constraints in its unbounded rectangular decomposition.

Lemma 54 For every diagonal free simple constraint C of the form $h \leq n$ or $h \geq n$, for every $M \in \mathbb{N}$, $[\![norm_M(C)]\!] = \cup_{C' \in Rect(C)} [\![norm_M(C')]\!]$.

Proof
If C is of the form:

- $h \leq n$,
  - If $M \geq n$ then $norm_M(C) = C$ and for every $C' \in Rect(C)$, $norm_M(C') = C'$. Then we get the result.
  - If $M < n$ then $norm_M(C) = tt$. Let C' be $h = n$. From Lemma 52, we get that $C' \in Rect(C)$ and $norm_M(C') = tt$; then we get that $\bigcup_{C' \in Rect(C)} [\![norm_M(C')]\!] = tt$ and $[\![norm_M(C)]\!] = \cup_{C' \in Rect(C)} [\![norm_M(C')]\!]$.
- $h \geq n$,
  - The case when $M \geq n$ is obvious because every constraint in $Rect(C) \cup \{C\}$ is not modified by $norm_M$.
  - The case when $M < n$ is also obvious because $norm(C) = h > M$ and $norm_M(C') = h > M$ for every $C' \in Rect(C)$

□

Now we can easily extend results in the lemma above to diagonal free constraints.

Proposition 55 For every diagonal-free constraint g, for every $M \in \mathbb{N}$, $[\![norm_M(g)]\!] = \bigcup_{\hat{g} \in Rect(g)} [\![norm_M(\hat{g})]\!]$.

Proof
It is a consequence of Lemma 54 above and Proposition 53 □

Definition 56 Given $g \in Gds_{\mathcal{H}}$ and an integer $M \in \mathbb{N}$, we define the set

$$Rect_M(g) = \{norm_M(\hat{g}) \mid \hat{g} \in Rect(g)\}$$

From Proposition 51, we get that every diagonal-free constraint using constant smaller than an integer M can be decomposed into a finite set of M-rectangular constraints.

Proposition 57 For every constraint $g \in Gds_{\mathcal{H}}$, for every $M \in \mathbb{N}$ such that $M \geq M_g$ we have that: $[\![g]\!] = \bigcup_{\hat{g} \in Rect_M(g)} [\![\hat{g}]\!]$.

Proof
From Proposition 55 $[\![norm_M(g)]\!] = \bigcup_{\hat{g} \in Rect(g)} [\![norm_M(\hat{g})]\!]$ or equivalently $[\![norm_M(g)]\!] = \bigcup_{\hat{g} \in Rect_M(g)} [\![\hat{g}]\!]$. From Proposition 51 for $M \geq M_g$, $[\![g]\!] = [\![norm_M(g)]\!]$ and we get the result. □

Remark: The same kind of property can be established for general constraints and triangular constraints. The semantics of every rectangular constraint is equal to the union of semantics of some triangular constraints. Then, every M-bounded diagonal free atomic constraint can be decomposed into an equivalent set of M-bounded triangular constraints.

From the remark above we have the following proposition that we leave without proof.

Proposition 58 Every constraint or diagonal free constraint can be decomposed into a finite equivalent set of triangular constraints.

2.2 Regions

We present a partitioning of the valuations into a finite number of equivalence classes called *regions*. Valuations in the same region must satisfy the same clock constraints, their time successors must also satisfy the same clock constraints, and they must satisfy the same clock constraints after a clock is reset. Depending on the nature of the clocks constraints, region are defined differently but they agree on a same set of properties.

Given a valuation v, $[v]$ denotes the equivalence class (region) of v. We also use the letter r to represent a region. Given a region r, we define $r + t = \{[v + t] \mid v \in r\}$, $r\uparrow = \{r + t \mid t \in \mathbb{R}_{\geq 0}\}$ and $r[h := 0] = \{[v[h := 0]] : v \in r\}$.

The operation $r + t$ returns the set of regions that can be reached from valuations in r after t time units. The operation $r\uparrow$ gives the set of regions that can be reached when the time elapses in r. The operation $r[h := 0]$ gives the unique region after the clock h is reset in every valuation of r. We write $r \subseteq g$ for $r \subseteq [\![g]\!]$.

Given a set G of constraints, we will present constructions for different types of sets of regions $\mathcal{R}eg$ that satisfy the following properties:

P1 $\forall g \in G, r \in \mathcal{R}eg$, either $r \subseteq [\![g]\!]$ or $[\![g]\!] \cap r = \emptyset$.

P2 $\forall r, r' \in \mathcal{R}eg$, if there exists some $v \in r$ and $t \in \mathbb{R}_{\geq 0}$ such that $v + t \in r'$, then for every $v' \in r$ there is some $t' \in \mathbb{R}_{\geq 0}$ such that $v' + t' \in r'$.

P3 $\forall r, r' \in \mathcal{R}eg, \forall h \in \mathcal{H}$, if $r[h := 0] \cap r' \neq \emptyset$, then $r[h := 0] \subseteq r'$.

Now we will present definitions of regions for diagonal-free constraints and general (diagonal) constraint.

2.2.1 Regions for Diagonal Free Constraints

The definition of a region we present here has been introduced by Alur and Dill [AD94] for analysing timed automata using only diagonal -free constraints. The equivalence relation between valuations is defined with respect to some integer M representing the maximal value used in constraints. The definition of that relation is somehow related to the definition of atomic constraints as atomic constraints can not be decomposed into smaller constraints. Thus, two equivalent valuations agree on the integral part of each clock whose values are smaller than M and they also agree on the order on the fractional part of the values of the clocks.

For a real number n let $\lfloor n \rfloor$ denote the integral part of n and $\{n\}$ denote the fractional part of n.

Let M be a natural number. Consider the parametrised binary relation $\sim^M \subseteq \mathcal{V}_{\mathcal{H}} \times \mathcal{V}_{\mathcal{H}}$ over valuations defined by, $v \sim^M v'$ if:

1. $v(h) > M$ if and only if $v'(h) > M$ for each $h \in \mathcal{H}$;
2. if $v(h) \leq M$, then $\lfloor v(h) \rfloor = \lfloor v'(h) \rfloor$ for every $h \in \mathcal{H}$;
3. if $v(h) \leq M$, then $\{v(h)\} = 0$ if and only if $\{v'(h)\} = 0$ for every $h \in \mathcal{H}$, and;
4. if $v(h) \leq M$ and $v(h') \leq M$, then $\{v(h)\} \leq \{v(h')\}$ if and only if $\{v'(h)\} \leq \{v'(h')\}$ for every $h, h' \in \mathcal{H}$.

Proposition 59 ([AD94]) The relation $\sim^M$ is an equivalence relation over the set of valuations with at most $2^{3|\mathcal{H}|-1} \times |\mathcal{H}|! \times (M+1)^{|\mathcal{H}|}$ equivalence classes.

Proof
The relation $\sim^M$ is defined as a conjunction of four properties. Each property defines an equivalence relation; let us denote them by $\sim_1^M, \ldots, \sim_4^M$, respectively. For each of these four relations we will give an upper bound on the number of its equivalence classes. The product of these bounds will give an upper bound on $\sim^M$ as the later is the intersection of the four equivalence relations.

The relation defined by the first condition has $2^{|\mathcal{H}|}$ equivalence classes, as the only thing that counts is whether the value of a clock is bigger than M or not. Similarly the third relation

has $2^{|\mathcal{H}|}$ equivalence classes. The number of classes of the second relation is $(M+1)^{|\mathcal{H}|}$ as there are $M+1$ possible integer values of interest. Finally, the number of classes of the fourth relation is bounded by the number of permutations of the set of clocks multiplied by $2^{|\mathcal{H}|-1}$ as for every two clocks consecutive in a permutation we need to decide if they are equal or if the second is strictly bigger than the first.

Summarizing, we get $2^{3|\mathcal{H}|-1}|\mathcal{H}!|(M+1)^{|\mathcal{H}|}$.

□

We use $\mathcal{R}eg(M)$ (or $\mathcal{R}eg$ for short) to represent the set of equivalence classes of the relation $\sim^M$.

Definition 60 A *region* [AD94] is an equivalence class of the relation $\sim^M \subseteq \mathcal{V}_\mathcal{H} \times \mathcal{V}_\mathcal{H}$ defined above.

In Figure 4 we illustrate region for diagonal free constraints for the maximal constant $M = 2$. In Figure 4 valuations earlier presented in Table 2 are not equivalent. A region in the figure is either a corner point (for example $(0, 2)$), an open line segment (for example $0 < h_1 = h_2 < 1$) or an open box (for example $0 < h_1 < h_2 < 1$).

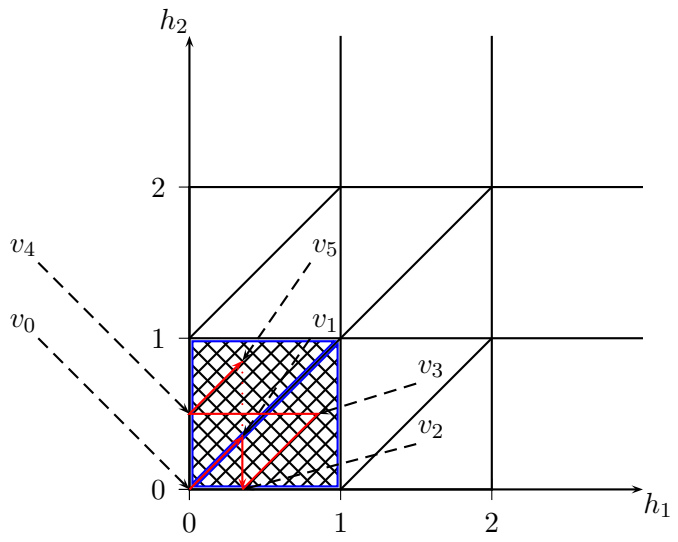

Figure 4: Region illustration.

From the definition of $\sim^M$, it comes that an equivalence class can be represented using a triangular constraint in g. According to the definition of $\sim^M$, two valuations that belong to the same equivalence class satisfy constraint of the form:

- $h = i_h$ or $i_h < h < i_h + 1$ for each $h \in H$ where $i_h \in \{0, 1, \ldots, M\}$ and we assume $M + 1 = \infty$. This is a consequence of $\sim_1^M$, $\sim_2^M$, $\sim_3^M$.
- $h - h' = i_{hh'}$ or $i_{hh'} < h - h' < i_{hh'} + 1$ for each couple $(h, h') \in H^2$ such that $h \bowtie M$ and $h' \bowtie M$ with $\bowtie \in \{=, <\}$. This is a consequence of $\sim_4^M$.

Proposition 61 Let G be a set of M-bounded constraints then $\mathcal{R}eg(M)$ satisfies the P1, P2, P3 mentioned above.

Proof
We show P1 in the first item, P2 in the second item and P3 in the last item.

1. Let $g \in G$, from Proposition 57 let $[\![g]\!] = \bigcup_{g_i \in Rect_M(g)} [\![\hat{g}_i]\!]$. Each $\hat{g}_i$ is a rectangular constraint. $[\![g]\!] \cap r = \bigcup_{g_i \in Rect_M(g)} [\![\hat{g}_i]\!] \cap r)$. From Fact 48 there is at most one i such that r intersects $\hat{g}_i$. It follows that r intersects a constraint $\hat{g}_i$ of $Rect_M(g)$ if and only if $\hat{g}_i$ contains r. We have that if $v \vDash r$ then $v \vDash g$.

2. Let $v, v' \in r$, adding t to v may modify the integer part of the value (with respect to v) of some clocks or may modify the order on the fractional part of the value (with respect to v) of clocks. We aim at find a time t' such that:

   - The integer part of the value of each clock with respect to $v' + t'$ is equal to the integer part of the value of each clock with respect to $v + t$
   - The order of the fractional parts of clocks in $v' + t'$ is the same in $v + t$.
   - The set of clocks with zero fractional part in $v + t$ is the same in $v' + t'$.

   Let $|\mathcal{H}| = n$ and assume a permutation π of $\{1, \ldots, n\}$ such that:

   $$\{v(h_{\pi_1})\} \bowtie_1 \{v(h_{\pi_2})\} \bowtie_2, \ldots, \bowtie_{n-1} \{v(h_{\pi_n})\} (*)$$

   with $\bowtie_i \in \{<, =\}$.

   Let $t \in \mathbb{R}_{\geq 0}$. It is clear that $\{v(h) + t\} = \{v(h) + \{t\}\}$. Only the fractional part of t may affect the order in $(*)$.

   There may be a largest index j such that:
   $\{v(h_{\pi_j}) + \{t\}\} = \{v(h_{\pi_j})\} + \{t\}$. In case, no such j exists, take $j = n$.

   Clearly, $\{v(h_{\pi_j}) + \{t\}\} \geq \{v(h_{\pi_j})\}$ and; $\forall k > j$ we have:
   $\{v(h_{\pi_k}) + \{t\}\} < \{v(h_{\pi_k})\}$ and $\{v(h_{\pi_k}) + \{t\}\} < \{v(h_{\pi_j}) + \{t\}\}$.

   We get that:

   $$\{v(h_{\pi_{j+1}}) + \{t\}\} \bowtie_j \ldots\ldots \bowtie_{n-1} \{v(h_{\pi_n}) + \{t\}\} < \{v(h_{\pi_j}) + \{t\}\}$$

   Similarly, we establish that

   $$\{v(h_{\pi_j}) + \{t\}\} < \{v(h_{\pi_{j-1}}) + \{t\}\} \overline{\bowtie}_{j-2} \ldots \overline{\bowtie}_1 \{v(h_{\pi_1}) + \{t\}\}$$

   where $\overline{\bowtie}_k$ => if $\bowtie_j \in \{<\}$ otherwise $\bowtie_j \in \{=\}$, $\forall k \leq j$

   - If $\{v'(h_{\pi_{j+1}}) + \{t'\}\} \neq 0$, in order to have

     $$\{v'(h_{\pi_{j+1}}\{t'\}\} \bowtie_j \ldots \bowtie_{n-1} < \{v'(h_{\pi_n}\{t'\}\} < \{v'(h_{\pi_j}) + \{t'\}\} \text{ and}$$

     $$\{v'(h_{\pi_{j+1}}) + \{t'\}\} \bowtie_j < \{v'(h_{\pi_{j-1}}) + \{t'\}\} \overline{\bowtie}_{j-2} \ldots \overline{\bowtie}_1 \{v'(h_{\pi_1}) + \{t'\}\}$$

     We take $\{t'\} \in [0, 1 - \{v'(h_{\pi_j})\}[\cap[1 - \{v'(h_{\pi_{j+1}})\}, 1[$.

- If $\{v'(h_{\pi_{j+1}}) + \{t'\}\} = 0$ then $\{t\} = 1 - \{v(h_{\pi_{j+1}})\}$; and we take $\{t'\} = 1 - \{v'(h_{\pi_{j+1}})\}$.

It comes that $\lfloor\{v'(h_{\pi_i})\} + \{t'\}\rfloor = \lfloor\{v(h_{\pi_i})\} + \{t\}\rfloor$.

To ensure that $\lfloor\{v'(h_{\pi_i})\} + t'\rfloor = \lfloor\{v(h_{\pi_i})\} + t\rfloor$ we must take $\lfloor t\rfloor = \lfloor t'\rfloor$.

3. Let $v_1, v_2 \in r$, then v_1 and v_2 satisfy all the conditions in the definition of an equivalence class. Its obvious that $v_1[h := 0]$ and $v_2[h := 0]$ also satisfy those three conditions and then $v_1[h := 0]$ and $v_2[h := 0]$ belong to $r[h := 0]$.

   If $v \in r[h := 0] \cap r'$ then every $v' \in r'$ is equivalent to v which is also equivalent to every $v'' \in r[h := 0]$. Thus $v \in r[h := 0]$ if and only if $v \in r[h := 0]$.

□

2.2.2 Regions for General Constraints

For general constraints, the ones in $\Phi_{\mathcal{H}}$, we need to slightly modify the equivalence relation defined above. Considering a set G of clock constraints, we consider the equivalence relation $\sim_G$ defined in [Yov98] by $v \sim_G v'$ if the following three conditions hold:

1. $v(h) > M$ implies $v'(h) > M$ where M is the maximal constant that occurs in G
2. if $v(h) \leq M$ then
   (a) $\lfloor v(h)\rfloor = \lfloor v'(h)\rfloor$ and
   (b) $\{v(h)\} = 0$ implies $\{v'(h)\} = 0$
3. For every clock constraints in G of the form $h - h' \sim c$, $v \vDash h - h' \sim c$ implies $v' \vDash h - h' \sim c$.

The set of regions for a set of clock constraints G is the set of equivalence classes of the relation $\sim_G$. Using similar argumentation as in the previous subsection, we can show that the number of equivalence classes is finite and the set of regions satisfies the properties P1, P2, P3 mentioned above.

2.3 Zones and Difference Bounded Matrices

2.3.1 Zone and Representation

We define zones that we use later for symbolic analysis. Zones have been considered by several authors [Dil90, HNSY94, YL97] for analysing timed systems. They enable a finite partitioning of valuations.

Definition 62 A *zone* is the set of valuations satisfying a constraint. A k-bounded zone is a zone defined by a k-bounded clock constraint.

For a zone Z represented by the constraint g, we define the approximation operator $norm_M(Z) = norm_M(g)$

Given a zone Z, the set of k-bounded zones containing Z is finite and not empty. The intersection of theses k-bounded zones is a k-bounded zone containing Z, and is the smallest one having this property.

Since it is obvious that several constraints represent the same zone, among the set of constraints with the same semantics we are interested in the so-called unique *canonical constraint*, the one that we can not strengthen any simple constraint it contains without modifying of the semantics of the constraint.

A zone can be represented with a difference bounded matrix [Dil90] defined below.

Definition 63 A difference bounded matrix (DBM) for n clocks is an $(n+1)\times(n+1)$ square matrix of pairs

$$(c;\bowtie) \in (\mathbb{Z} \times \{<,\leq\}) \cup \{(\infty;<)\}$$

A DBM $\mathcal{D} = (c_{i,j}, \bowtie_{i,j})_{i,j=1..n}$ defines the following subset of valuations (the clock h_0 is supposed to be always equal to zero, that is, for each valuation v, $v(h_0) = 0$):

$$\{v : \{h_1, h_2, \ldots, h_n\} \to \mathbb{R}^+ \mid \forall 0 < i, j < n, v(h_i) - v(h_j) \bowtie_{i,j} c_{i,j}\}$$

We will write $v \in \mathcal{D}$ if the valuation v belong to the set that define $\mathcal{D}$.

It is obvious that a DBM $\mathcal{D}$ can be translated into a constraint. That constraint is just the conjunction of simple constraints of the form $h_i - h_j \bowtie_{i,j} c_{i,j}$. A DBM is canonical if the constraint associated to it is canonical.

2.3.2 Computation of some Operations on DBMs

We recall operations on DBMs that are useful for the reachability analysis. These operations have been nicely described in [BY04, CGP99]. Some of these operations use the following order relation and the sum (+) operation between elements of DBMs. In Subsection 2.6.2, these operations are used to compute some DBMs (presented there in a simple form).

Given $(c;\bowtie)$ and $(c';\bowtie')$ two possible elements of a DBMs, we define the order $\leq_e \subseteq ((\mathbb{Z}\times\{<,\leq\}) \cup \{(\infty;<)\})^2$ by

$$(c;\bowtie) \leq_e (c';\bowtie') \Longrightarrow \begin{cases} c < c' \\ or \\ c = c' \quad \text{and either} \quad \bowtie = \bowtie' \quad or \quad \bowtie' = \leq \end{cases}$$

We define $(c;\bowtie) + (c';\bowtie') = (c'';\bowtie'')$ where $c'' = c + c'$ and $\bowtie''$ is $\leq$ if both $\bowtie$ and $\bowtie'$ are $\leq$ and $\bowtie''$ is $<$ otherwise.

Canonical Dbms: The computation of canonical Dbms derives tightest simple diagonal constraints, one for each simple diagonal constraint in Dbms. A given Dbm is transformed into a weighted graph where clocks are nodes and the simple constraints are edges labeled with bounds. A constraints of the form $h - h' \bowtie n$ (with $\bowtie \in \{<,\leq\}$) will be converted to an

edge from node h' to h labeled with $(\bowtie, n)$, namely the distance from h' to h is bounded by n. Thus, deriving the tightest simple constraint on a pair of clocks in a Dbm, is equivalent to find the shortest path between their nodes in the weighted graph constructed from the Dbm. The Floyd-Warshall algorithm [Flo62] can be used to find shortest paths between nodes. There is a simple constraint $h - h' \bowtie n'$ in the resulting canonical Dbm if the weight of the shortest path from h' to h is equal to $(\bowtie, n')$.

Intersection: Let $\mathcal{D} = (c_{i,j}; \bowtie_{i,j})_{i,j=1,\ldots,n}$ and $\mathcal{D}' = (c'_{i,j}; \bowtie'_{i,j})_{i,j=1,\ldots,n}$ be two DBMs. Consider the DBM $\mathcal{D}'' = (c''_{i,j}; \bowtie''_{i,j})_{i,j=1,\ldots,n}$ defined by:

$$(c''_{i,j}; \bowtie''_{i,j}) = min((c_{i,j}; \bowtie_{i,j}), (c'_{i,j}; \bowtie'_{i,j}))\text{for all indexes } i, j = 1 \ldots n$$

where $min(x, y)$ denotes the minimum of x and y according to the relation $\leq_e$ defined above. It has been established that $v \in \mathcal{D}''$ if and only if $v \in \mathcal{D}$ and $v \in \mathcal{D}'$

Future: This operation computes the set of valuations that are reachable from a DBM when time elapses. Given $\mathcal{D} = (c_{i,j}; \bowtie_{i,j})_{i,j=1,\ldots,n}$ in normal form, the DBM $\mathcal{D}' = (c'_{i,j}; \bowtie'_{i,j})_{i,j=1,\ldots,n}$ defined by :

$$\begin{cases} (c'_{i,j}; \bowtie'_{i,j}) & = & (c_{i,j}; \bowtie_{i,j}) & if & j \neq 0 \\ (c'_{i,j}; \bowtie'_{i,j}) & = & (\infty; <) \end{cases}$$

is such that $v' \in \mathcal{D}'$ if and only if there is $t \in \mathbb{R}^+$, $v \in \mathcal{D}$ such that $v' = v + t$.

Past: This operation computes the set of valuations from which a valuation in a DBM can be reached when time elapses. Given $\mathcal{D} = (c_{i,j}; \bowtie_{i,j})_{i,j=1,\ldots,n}$ in normal form, the DBM $\mathcal{D}' = (c'_{i,j}; \bowtie'_{i,j})_{i,j=1,\ldots,n}$ defined by :

$$\begin{cases} (c'_{i,j}; \bowtie'_{i,j}) & = & (c_{i,j}; \bowtie_{i,j}) & if & i \neq 0 \\ (c'_{i,j}; \bowtie'_{i,j}) & = & (0; \leq) & if & i = 0 \end{cases}$$

is such that $v' \in \mathcal{D}'$ if and only if there is $t \in \mathbb{R}^+$, $v \in \mathcal{D}$ such that $v = v' + t$.

Image by resets: Assume that $\mathcal{D} = (c_{i,j}; \bowtie_{i,j})_{i,j=1,\ldots,n}$ is a DBM in canonical form. Consider the DBM $\mathcal{D}_{h_k:=0} = (c'_{i,j}; \bowtie'_{i,j})_{i,j=1,\ldots,n}$ defined by:

$$\begin{cases} (c'_{i,j}; \bowtie'_{i,j}) & = & (c_{i,j}; \bowtie_{i,j}) & if & i, j \neq k \\ (c'_{k,k}; \bowtie'_{i,j}) & = & (c'_{k,0}; \bowtie'_{k,0}) & = & c'_{0,k}; \bowtie'_{0,k}) & = (0; \leq) \\ (c'_{i,k}; \bowtie'_{i,k}) & = & (c_{i,0}; \bowtie_{i,0}) & & if & i \neq k \\ (c'_{k,i}; \bowtie'_{k,i}) & = & (c_{0,i}; \bowtie_{0,i}) & & if & i \neq k \end{cases}$$

We have that $v' \in \mathcal{D}_{h_k:=0}$ if and only if there is $v \in \mathcal{D}$ such that $v' = v[h_k := 0]$.

k-approximation: The k- approximation of a DBM $\mathcal{D} = (c_{i,j}; \bowtie_{i,j})_{i,j=1,\ldots,n}$ in canonical form is the DBM $\mathcal{D}_k = (c'_{i,j}; \bowtie'_{i,j})_{i,j=1,\ldots,n}$ defined by:

$$\begin{cases} (c'_{i,j}; \bowtie'_{i,j}) & = & (c_{i,j}; \bowtie_{i,j}) & if & -k \leq c_{i,j} \leq k \\ (c'_{i,j}; \bowtie'_{i,j}) & = & (\infty; <) & if & c_{i,j} > k \\ (c'_{i,j}; \bowtie'_{i,j}) & = & (-k; <) & if & c_{i,j} \leq -k \end{cases}$$

If Z is the zone associated to $\mathcal{D}$ then, the zone of $\mathcal{D}_k$ is equal to $Norm_k(Z)$

Emptiness testing: A DBM $\mathcal{D} = (c_{i,j}; \bowtie_{i,j})_{i,j=1,\dots,n}$ is empty if and only if there exists a negative cycle in $\mathcal{D}$, that means that there exists a sequence of distinct indexes $(i_1, i_2, \dots, i_{l-1}, i_l = i_1)$ such that

$$\sum_{j=1}^{l-1} (c_{i_j,i_{j+1}}, \bowtie_{i_j,i_{j+1}}) < (0; \leq)$$

Equality testing $\mathcal{D} = (c_{i,j}; \bowtie_{i,j})_{i,j=1,\dots,n}$ and $\mathcal{D}' = (c'_{i,j}; \bowtie'_{i,j})_{i,j=1,\dots,n}$ be two DBMs assumed to be in canonical form. $\mathcal{D}$ is equal to $\mathcal{D}'$ if and only if $(c_{i,j}; \bowtie_{i,j}) \leq_e (c'_{i,j}; \bowtie'_{i,j})$ and $(c'_{i,j}; \bowtie'_{i,j}) \leq (c_{i,j}; \bowtie_{i,j})$

2.4 Models for Timed Processes

2.4.1 Definitions

We present models for timed processes, their semantics and two alternative representations for the semantics. Models for timed process are nothing else but event-recording automata [AFH99] without an acceptance condition. The semantics of timed processes are transition systems. States of semantics are couple of the form (p, v) where p is a (control) state of a timed process and the valuation v gives the timing context of the execution of the timed process. Transitions of semantics are of two sorts: *delay transitions* that represent the elapse of the time and *discrete transitions* that represent the occurrence of an event. Our presentation for Delay transition is not standard. Instead of labeling delay transition with real number t as it is commonly done in [AD94, AFH99, DM02, BCL05] to represent the elapse of time from the configuration (p, v), we choose to label delay transitions with the timing context $v + t$ from (p, v) after t time units. The two representations for the semantics successively replace valuations on transitions with atomic constraints and valuation in configuration with regions.

Let $\Sigma = \{a_1, a_2 \dots\}$ be a set of *events* . We consider $\mathcal{H}_\Sigma = \{h_1, h_2, \dots\}$ the set of clocks. The clock h_i is the unique clock associated to the event a_i. When there is no confusion, a will denote an event and h_a will denote the unique clock associated to a. There are as many clocks as events. The symbol Gds_Σ will denote the set of constraints defined over $\mathcal{H}_\Sigma$, the symbol $Agds_\Sigma$ will denote the set of rectangular constraints over $\mathcal{H}_\Sigma$, and the symbol $\mathcal{V}_\Sigma$ will denote the set of valuations over $\mathcal{H}_\Sigma$.

Definition 64 A *timed process* , or process for short, is a tuple

$$\mathcal{P} = \langle P, \Sigma, p^0, \Delta_P \rangle$$

where,

- P is a finite set of states,
- $p^0 \in P$ is the initial state,

- $\Delta_P \subseteq P \times Gds_\Sigma \times \Sigma \times P$ is a transition relation.

Sometimes, we shortly write $p \xrightarrow{g,a} p'$ for a transition (p, g, a, p') in Δ_P. The *bound* of a timed process is the maximal constant that occurs in its guards. For a timed process $\mathcal{P}$, $M_\mathcal{P}$ denotes its bound. Given a constant M, we say a timed process is M-bounded if its bound is smaller than M.

Definition 65 A timed process is *deterministic* if whenever there are two transitions $p \xrightarrow{g_1,a} p_1$ and $p \xrightarrow{g_2,a} p_2$ with $p_1 \neq p_2$, the constraint $g_1 \wedge g_2$ is inconsistent.

Figure 5,Figure 6 and Figure 7 present three timed processes. The timed process in Figure 6 and Figure 7 are deterministic and timed process in Figure 5 is not deterministic.

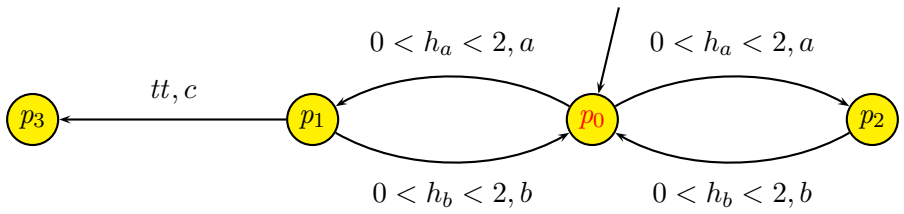

Figure 5: A non deterministic timed process:$\mathcal{P}_0$.

The timed process in Figure 5 is not deterministic as the conjunction of the guards in the two transitions outgoing from p_0 is consistent while their events are the same. In Figure 6, the conjunction of the guards is inconsistent and, in Figure 7 the transitions outgoing from p_0 are not labelled with the same event.

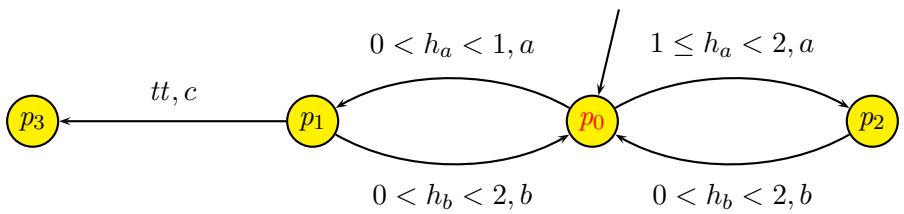

Figure 6: A deterministic timed process:$\mathcal{P}_1$.

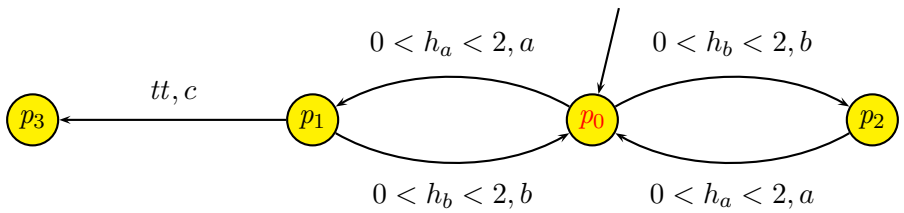

Figure 7: A deterministic timed process:$\mathcal{P}_2$.

2.4.2 Semantics

The semantics of a timed process is a transition system that represents all possible behaviours of the timed process. The idea is that each clock h_a records the amount of time elapsed since the last occurrence of the corresponding event a. The time elapses continuously at a state. Whenever an action a is executed, the clock h_a is automatically reset. No other clock assignments are permitted.

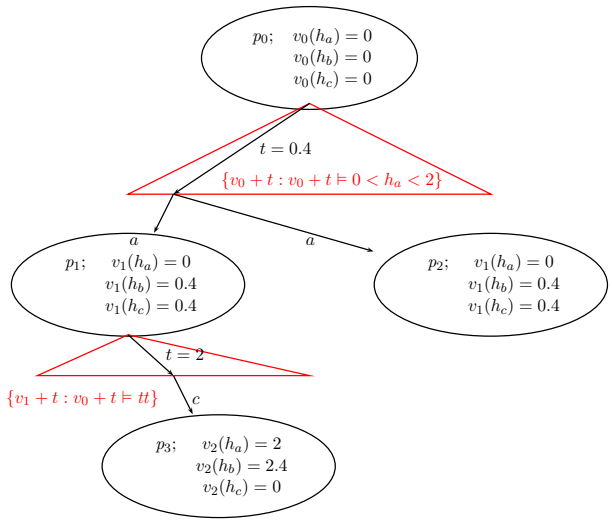

Figure 8: A part of the semantics of $\mathcal{P}_0$.

Definition 66 The semantics of a timed process $\mathcal{P}$ as above is the transition system

$$[\![\mathcal{P}]\!] = \langle P \times \mathcal{V}_\Sigma, \Sigma \cup \mathcal{V}_\Sigma, (p^0, v^0), \rightarrow \rangle$$

where $\rightarrow \subseteq (P \times \mathcal{V}_\Sigma) \times (\Sigma \cup \mathcal{V}_\Sigma) \times (P \times \mathcal{V}_\Sigma)$ is defined by:

- $(p, v) \xrightarrow{v+t} (p, v + t)$ for every $t \geq 0$.
- $(p, v) \xrightarrow{a} (p', v[h_a := 0])$ if there is $(p, g, a, p') \in \Delta_P$ such that $v \in [\![g]\!]$.

Delay transitions are transitions labelled with valuations and *discrete transitions* are transitions labelled with events.

Remark: When presenting the semantics of timed automata [AD94, DM02, BCL05] and event-recording automata [AFH99], it is usual to label delay transitions with non negative real numbers. In the semantics presented above, delay transition are labelled with valuations. We remark that these two presentations are equivalent. The choice of the presentation above will be justified in the next chapters when the semantics of formulas will be defined.

Notation: Later we use the notation $s \xrightarrow{v,a} s'$ if there exists s'' such that $s \xrightarrow{v} s''$ and $s'' \xrightarrow{a} s'$.

Let us use the following example to illustrate the notion of semantics of timed processes. We consider process in Figure 5 and Figure 6 and transitions from p_0 to p_1 and p_2. In Figure 8, we present the beginning of the semantics of the process in Figure 5. As that process is not deterministic, at the same time (for example $t = 0.4$), it is possible to trigger the event a and either move to p_1 or p_2. From p_1 it is possible to do immediately c while it is not the case from p_2.

2.4.3 Representations for Timed Processes

The above semantics is not very convenient as both the set of states and the set of labels occurring in transitions are uncountable. We will consider two more abstract semantics of processes. The first will abstract from valuations in the labels of transitions. The second will replace valuations in states by regions. In order for the abstractions to be finite, they will be parametrized by a bound M on the clock values.

Definition 67 The *M-action abstraction* of a timed process $\mathcal{P}$ is the $(\Sigma \cup Agds_\Sigma(M))$-labeled transition system

$$\langle\!|\mathcal{P}|\!\rangle^M = \langle P \times \mathcal{V}_\Sigma, \Sigma \cup Agds_\Sigma(M), (s^0, v^0), \Delta_v \rangle,$$

where $\Delta_v \subseteq (P \times \mathcal{V}_\Sigma) \times (\Sigma \cup Agds_\Sigma(M)) \times (P \times \mathcal{V}_\Sigma)$ is defined by:

- $(p, v) \xrightarrow{\hat{g}} (p, v + t)$ for any $t \in \mathbb{R}^+$ such that $v + t \vDash \hat{g}$ and
- $(p, v) \xrightarrow{a} (p', v[h_a := 0])$ if there is $(p, g, a, p') \in \Delta_P$ with $v \vDash g$.

We observe that the M-action representation is obtained from the semantics by replacing valuations on delay-transitions with M-rectangular constraints they satisfy. Then for every timed process $\mathcal{P}$ and every natural constant M, there is an isomorphism between $[\![\mathcal{P}]\!]$ and $\langle\!|\mathcal{P}|\!\rangle^M$.

Definition 68 The *M-region abstraction* of a timed process $\mathcal{P}$ is the $(\Sigma \cup Agds_\Sigma(M))$-labeled transition system

$$\langle\!|\mathcal{P}|\!\rangle^M_{reg} = \langle P \times \mathcal{R}eg(M), \Sigma \cup Agds(M), (p^0, r^0), \Delta_r \rangle,$$

where $v^0 \in r^0$, $\Delta_r \subseteq (P \times \mathcal{R}eg(M)) \times (\Sigma \cup Agds_\Sigma(M)) \times (P \times \mathcal{R}eg(M))$ is defined by:

- $(p, r) \xrightarrow{\hat{g}} (p, r')$ with $r' \subseteq r\uparrow$ and $r' \subseteq \hat{g}$.
- $(p, r) \xrightarrow{a} (p', r[h_a := 0])$ if there is $(p, g, a, p') \in \Delta_P$ with $r \subseteq g$.

Notation: Later and particularly in the next Chapter, given a $(Gds_\Sigma \cup \Sigma)$-LTS, we use the notation $s \xrightarrow{g,a} s'$ if there exists s'' such that $s \xrightarrow{g} s''$ and $s'' \xrightarrow{a} s'$.

Proposition 69 For every timed process $\mathcal{P}$, and every $M \geq M_\mathcal{P}$: $\langle\!|\mathcal{P}|\!\rangle^M$ is bisimilar to $\langle\!|\mathcal{P}|\!\rangle^M_{reg}$.

Proof
We consider a relation $\sim \subseteq (P \times \mathcal{V}_\Sigma) \times (P \times \mathcal{R}eg_\Sigma(M))$ defined by $(p, v) \sim (p, [v])$ for every $p \in P, v \in \mathcal{V}_\Sigma$. We show that it is a bisimulation.

- First, we consider delay transitions. Assume that $(p, v) \sim (p, [v])$. If $(p, v) \xrightarrow{\hat{g}} (p, v')$, then there is $t \in \mathbb{R}^+$ such that $v + t \in [\![\hat{g}]\!]$. According to Proposition 61, $[v + t] \subseteq \hat{g}$ and obviously $[v+t] \subseteq [v]\uparrow$. Then, we get that $(p, [v]) \xrightarrow{\hat{g}} (p, [v+t])$ and $(p, v+t) \sim (p, [v+t])$. Reciprocally, if $(p, r) \xrightarrow{\hat{g}} (p, r')$, then $r' \subseteq \hat{g}$ and $r' \subseteq r\uparrow$. Let $v \in r$ according to Proposition 61, there is $t \in \mathbb{R}^+$ such that $v + t \in r'$. Since $r' \subseteq \hat{g}$, we get $v + t \in [\![\hat{g}]\!]$ and then $(p, v) \xrightarrow{\hat{g}} (p, v')$.
- Next, we consider discrete transitions. Assume that $(p, v) \sim (p, [v])$. If $(p, v) \xrightarrow{a} (p', v')$, then $v' = v[h_a := 0]$ and there is $p \xrightarrow{g,a} p'$ such that $v \in [\![g]\!]$. Let $\hat{g} \in Agds(M)$ be an atomic guard such that $v \in [\![\hat{g}]\!]$. Then we get $(p, [v]) \xrightarrow{a} (p', [v'])$ and $(p, v') \sim (p, [v'])$. Reciprocally, if $(p, r) \xrightarrow{a} (p', r')$, then $r' = r[h_a := 0]$ and there is $p \xrightarrow{g,a} p'$ such that $r \in [\![g]\!]$. Let $v \in r$, obviously $v \in [\![g]\!]$, and $v[h_a := 0] \in r'$. It follows that $(p, v) \xrightarrow{a} (p', v[h_a := 0])$ and $(p, v[h_a := 0]) \sim (p, r')$.

□

2.5 Product of Timed Processes

Definition 70 The *product* of a timed process $\mathcal{P} = \langle P, \Sigma, p^0, \Delta_P \rangle$ with a timed process $\mathcal{R} = \langle S, \Sigma, s^0, \Delta_\mathcal{R} \rangle$ is the timed process denoted by $\mathcal{P} \times \mathcal{R}$ and defined by the tuple $\mathcal{P} \times \mathcal{R} = \langle P \times S, \Sigma, (p^0, s^0), \Delta \rangle$ where $((p, s), g, a, (p', s')) \in \Delta$ if there is $(p, g', a, p') \in \Delta_P$, $(s, g'', a, s') \in \Delta_\mathcal{R}$ with $g = g' \wedge g''$.

Example: The product of the timed process in Figure 6 with the timed process in Figure 9 is depicted in Figure 10

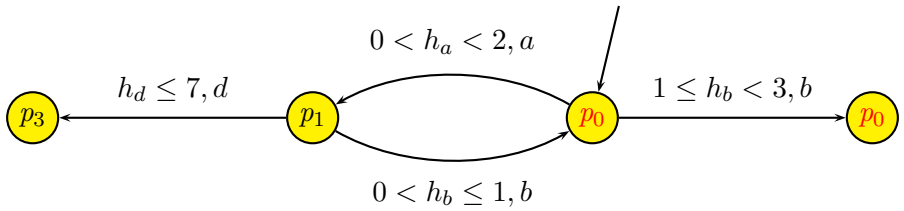

Figure 9: A non deterministic timed process:$\mathcal{P}_4$.

□

Now we show that the semantics of the product of two timed processes is the product of their semantics.

Lemma 71 Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two timed processes, then $[\![\mathcal{P}_1 \times \mathcal{P}_2]\!]$ is bisimilar to $[\![\mathcal{P}_1]\!] \times [\![\mathcal{P}_2]\!]$.

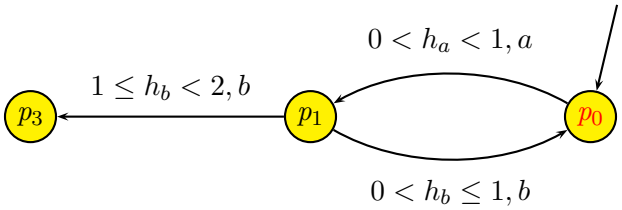

Figure 10: A non deterministic timed process:$\mathcal{P}_5$.

Proof
Let $\mathcal{P}_1 = \langle P_1, \Sigma, p_1^0, \Delta_{P1} \rangle$ and $\mathcal{P}_2 = \langle P_2, \Sigma, p_2^0, \Delta_{P2} \rangle$ and consider the relation

$$R \subseteq (P_1 \times P_2 \times \mathcal{V}_\Sigma) \times ((P_1 \times \mathcal{V}_\Sigma) \times (P_2 \times \mathcal{V}_\Sigma))$$

defined by $(p_1, p_2, v)R((p_1, v), (p_2, v))$. Now we show that R is a bisimulation between $[\![\mathcal{P}_1 \times \mathcal{P}_2]\!]$ and $[\![\mathcal{P}_1]\!] \times [\![\mathcal{P}_2]\!]$.

- It is obvious that $(p_1, p_2, v) \xrightarrow{v'} (p_1', p_2', v')$ is and only if $((p_1, v), (p_2, v)) \xrightarrow{v'} ((p_1', v'), (p_2', v'))$
- Let us consider discrete transitions. Assume that $(p_1, p_2, v)R((p_1, v), (p_2, v))$. If $(p_1, p_2, v) \xrightarrow{a} (p_1', p_2', v')$ then $v' = v[h_a := 0]$ and there is $(p_1, p_2) \xrightarrow{g,a} (p_1', p_2')$ such that $v \in [\![g]\!]$. But $(p_1, p_2) \xrightarrow{g,a} (p_1', p_2')$ implies there is $p_1 \xrightarrow{g_1,a} p_1'$ in $\mathcal{P}_1$ and $p_2 \xrightarrow{g_2,a} p_2'$ in $\mathcal{P}_2$ such that $[\![g]\!] = [\![g_1]\!] \cap [\![g_2]\!]$. As $v \in [\![g]\!]$, we get that $v \in [\![g_1]\!]$ and $v \in [\![g_2]\!]$ implying that $(p_1, v) \xrightarrow{a} (p_1', v')$ and $(p_2, v) \xrightarrow{a} (p_2', v')$ and then, we get that $((p_1, v), (p_2, v)) \xrightarrow{a} ((p_1', v'), (p_2', v'))$. Reciprocally if $((p_1, v), (p_2, v)) \xrightarrow{a} ((p_1', v'), (p_2', v'))$, then $(p_1, v) \xrightarrow{a} (p_1', v')$ and $(p_2, v) \xrightarrow{a} (p_2', v')$ with $v = v[h_a := 0]$. But $(p_i, v) \xrightarrow{a} (p_i', v')$ for every $i \in \{1, 2\}$ implies that there exists $p_i \xrightarrow{g_i,a} p_i'$ in $\mathcal{P}_i$ with $v \in [\![g_i]\!]$ for every $i \in \{1, 2\}$. It is obvious that $(p_1, p_2) \xrightarrow{g_1 \wedge g_2, a} (p_1', p_2')$ and because $v \in [\![g_1 \wedge g_2]\!]$ we get that $(p_1, p_2, v) \xrightarrow{a} (p_1', p_2', v')$.

□

We remark that the relation R in the proof above is a bijective application between states of $[\![\mathcal{P}_1 \times \mathcal{P}_2]\!]$ and $[\![\mathcal{P}_1]\!] \times [\![\mathcal{P}_2]\!]$. We get that $[\![\mathcal{P}_1 \times \mathcal{P}_2]\!]$ is isomorphic to $[\![\mathcal{P}_1]\!] \times [\![\mathcal{P}_2]\!]$.

2.6 Reachability Analysis

For verification purposes, the most fundamental properties that we should be able to verify are reachability properties. We consider the reachability analysis of timed processes. The reachability analysis requires to check whether a system has an execution from a given *start (or source)* state to a given *end (or target)* state. There are two main algorithms for the reachability analysis: the forward algorithm and the backward algorithm.

The forward analysis starts in a source state with clocks initialized with some set of values. Then, it computes states reachable within 1 steps, 2 steps, etc... until the target state is

reached or until the computation terminates. The backward algorithm starts in a target state with clocks initialized with some set of values and it computes states from which we can reach target states within 1 steps, 2 steps, etc... until source states are reached or until the computation terminates. As timing information needs to be considered for taking transitions in a timed process, simple algorithm may consider semantics of timed processes. Because of semantics of timed process are infinite state transitions systems where each state could have infinitely many successors and predecessors, an algorithm like depth in first search (DFS) may not terminate. Then, we need a finite representation for semantics that preserves reachability properties. States in that finite representation are pairs of a state of the timed process and a representable (infinite) set of set of valuations (that may be a singleton).

The forward algorithms are based on the computation of the representable set of valuations $Post(V, tr)$ of time successors of a representable set of valuations V with respect to a transition $tr = p \xrightarrow{g,a} p'$.

$$Post(V, tr) = \{v + t[h_a := 0], \mid v \in V \ \exists t \in \mathbb{R}^+ \text{ such that } v + t \in [\![g]\!]\}$$

Having $Post(V, tr)$, the forward analysis consists in computing the following symbolic states Src_i with $i \geq 0$. Src_0 is the symbolic start state made of a start state from the timed process and a set V^0 of initial valuations.

$$\begin{aligned} Src_0 &= \{(p, V^0) | p \text{ is the start state and } V^0 \text{ is an initial set of valuations}\} \\ \text{and iteratively} & \\ Src_{i+1} &= \{(p', V') | \exists tr = p \xrightarrow{g,a} p' \exists (p, V) \in Src_i \text{ such that } V' = Post(V, tr)\} \end{aligned}$$

The backward algorithms are based on the computation of the representable set of valuations $Pre(V, tr)$ of time predecessors of a representable set of valuations V with respect to a transition $tr = p \xrightarrow{g,a} p'$.

$$Pre(Z, tr) = \{v | \exists t \in \mathbb{R}^+ \text{ such that } v + t \in [\![g]\!] \text{ and } v + t[h_a := 0] \in Z\}$$

Then the backward analysis consists in computing the following symbolic states:

$$\begin{aligned} Tgt_0 &= \{(p, V^0) | p \text{ is the target state and } V^0 \text{ is an initial set of valuations}\} \\ \text{and iteratively} & \\ Tgt_{i+1} &= \{(p', V') | \exists tr = p \xrightarrow{g,a} p' \exists (p, V) \in Tgt_i \text{ such that } V' = Pre(V, tr)\} \end{aligned}$$

We will present algorithms that use regions and zones as representable set of valuations.

2.6.1 Region-based Algorithms

Region based algorithms have been introduced by Alur et Dill [AD94] for reachability analysis of timed automata. In region-based reachability algorithms, regions are used for representing sets of valuations. In this case, reachability algorithms work on M-region representations as in that representation states of timed processes are already paired with regions.

We explain here how one can use the M-region representation for reachability analysis of timed processes.

Lemma 72 For every M, a state is reachable in $[\![\mathcal{P}]\!]$ if and only if it is reachable in $(\!|\mathcal{P}|\!)^M$.

Proof
It is enough to remark that $(\!|\mathcal{P}|\!)^M$ is obtained from $[\![\mathcal{P}]\!]$ by renaming the labels of transitions. □

As $Agds_\Sigma(M)$ and Σ are finite, labels of transitions in $(\!|\mathcal{P}|\!)^M$ range over a finite set, but a state may also have infinitely many successors or predecessors. Then the following lemma comes as a corollary of Proposition 69. Let us recall that $M_\mathcal{P}$ is the maximal constant that occurs in the constraints of the timed process $\mathcal{P}$.

Lemma 73 For every $M \geq M_\mathcal{P}$, a state (p, v) is reachable in $[\![\mathcal{P}]\!]$ if and only if $(p, [v])$ is reachable in $(\!|\mathcal{P}|\!)^M_{reg}$

Proof
By Lemma 72 a state (p, v) is reachable in $[\![\mathcal{P}]\!]$ if and only if it is reachable in $(\!|\mathcal{P}|\!)^M$. Then we can use the bisimilarity result in Proposition 69 to conclude. □

In practice the region construction is not used to check reachability properties as the number of regions is too high. Algorithms for "minimizing" the region graph have been proposed for example in [ACD+92, ACH+92, TY01] and other techniques for "minimizing" reachability graph have been proposed for example in [YL97, KL96]. However in practice *on-the-fly* techniques are preferred since the reachability graph need not be entirely constructed before the analysis.

2.6.2 Zone-based Algorithms

The zone abstraction [LPY97] is a symbolic approach in which zones are used for representing a timing context.

Backward reachability algorithm

For a given $tr = p \xrightarrow{g,a} p'$ and a zone Z, the Pre operator defined above is specialised for zone as follows:

$$Pre(Z, tr) \quad = \quad \{v | \exists t \in \mathbb{R}^+ \text{ such that } v + t \in [\![g]\!] \text{ and } v + t[h_a := 0] \in Z\}$$

We recall the following result concerning the termination and the correctness of the algorithm that establishes that if a state is declared reachable by the computation, then it is really reachable. This result is just a corollary of a similar result on timed automata presented in [Bou03].

Proposition 74 The backward analysis algorithm terminates and is correct with respect to reachability.

Forward reachability algorithm

For a given $tr = p \xrightarrow{g,a} p'$ and a zone Z, the $Post$ operator defined above is specialised for zones as follows:

$$Post(Z, tr) = \{v + t[h_a := 0], \mid v \in Z \exists t \in \mathbb{R}^+ \text{ such that } v + t \in [\![g]\!]\}$$

But iterative computations of the Src_i will not terminates as $Post(Z, tr)$ may introduce new zones. For ensuring termination, approximation of zones has been proposed leading to the following algorithm.

Algorithm The following forward algorithm comes from [BY04, Bou03] and has been implemented in several tools like Kronos [BTY97, Yov98], and Uppaal [BLL+96, LPY97, BDL04].

Algorithm 1 Forward Analysis Algorithm for Timed processes

Require: $\mathcal{P} = \langle P, \Sigma, p^0, \Delta_P \rangle$
Require: $Target \subseteq P$ // the set of reachable state
Ensure: YES or NO a state of $Target$ is reachable.
  $Visited \leftarrow \emptyset$;
  $Waiting \leftarrow \{p^0, Norm_k(Z_0)\}$;
  repeat
    Get and Remove (p, Z) from $Waiting$;
    if $p \in Target$ **then**
      RETURN "YES, p is reachable";
    else
      if there is no $(q, Z') \in Visited$ such that $Z \subseteq Z'$ **then**
        $Visited \leftarrow Visited \cup \{(p, Z)\}$;
        $Successor \leftarrow \{(p', Norm_k(Post(Z, e) \mid e \text{ transition from } p \text{ to } p'\}$;
        $Waiting \leftarrow Waiting \cup Successor$;
      end if
    end if
  until $Waiting = \emptyset$
  RETURN NO;

Correctness Bouyer [Bou03](see also [BLR05]) has shown that the reachability algorithm is correct for timed automata that use only diagonal free constraints; but is not correct for timed automata that use general class of constraints. We show that the same is true for timed processes (that are special kinds of timed automata). For that purpose, we consider the timed process (we recall that only the clock associated to the event of a transition is reset when the transition is crossed) in Figure 11. This example is a very minor modification of the example from [Bou03].

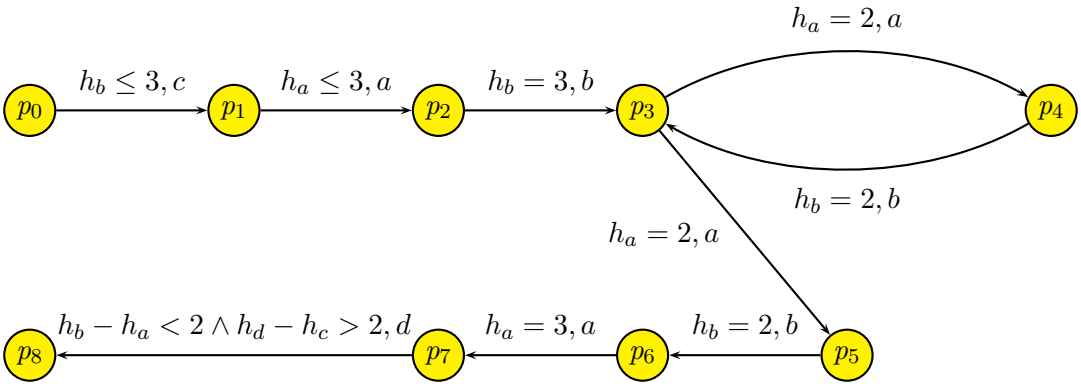

Figure 11: A timed process

We consider a path from p_0 to p_7 in the timed process in Figure 11. If α is the date the transition from p_0 to p_1 is taken, β is the date the transition from p_1 to p_2 is taken and γ is the the number of loops taken along the run, the valuation v of the clocks when arriving in p_7 is defined by:

$$\begin{array}{ll} v(h_a) = 0 & v(h_c) = 2\gamma + 5 + (\beta - \alpha) \\ v(h_b) = \beta & v(h_d) = 2\gamma + 5 + \beta \end{array}$$

Clearly, $v(h_d) - v(h_c) \leq v(h_b) - v(h_a)$ as $\beta \geq \alpha$.

If we consider the forward algorithm for reachability that uses zones and that starts in p_0 with all the clocks set to 0, the set of valuation that can be reached in p_7 when loop is taken γ times, is the following:

$$\begin{cases} h_a = 0 \\ h_b \geq 1 \\ h_c \geq 2\gamma + 5 \\ h_d \geq 2\gamma + 6 \\ 2\gamma + 6 \leq h_a - h_d \leq 2\gamma + 8 \\ 1 \leq h_b - h_a \leq 3 \\ 2\gamma + 5 \leq h_c - h_a \leq 2\gamma + 8 \\ 2\gamma + 2 \leq h_c - h_b \leq 2\gamma + 5 \\ h_d - h_b = 2\gamma + 5 \\ 0 \leq h_d - h_c \leq 3 \end{cases}$$

Consider the constraint $h_b - h_a < 2 \wedge h_d - h_c > 2$. If we require that $h_b - h_a < 2$ then,

$$\begin{array}{rcl} h_d - h_c & = & (h_d - h_b) + (h_b - h_a) + (h_a - h_c) \\ & \leq & -2\gamma - 5 + 2 + 2\gamma + 5 \\ & \leq & 2 \end{array}$$

This means that the transition from p_7 to p_8 can not be triggered from the zone above.

If we fix a maximal constant k and we use the approximation operator $Norm_k$, then for a number of loops γ sufficiently large and such that $k < 2\gamma + 2$, the approximated zone we obtain is the following:

$$\begin{cases} h_a = 0 \\ h_b \geq 1 \\ h_c > k \\ h_d > k \\ h_a - h_d > k \\ 1 \leq h_b - h_a \leq 3 \\ h_c - h_a > k \\ h_c - h_b > k \\ h_d - h_b > k \\ 0 \leq h_d - h_c \leq 3 \end{cases}$$

This zone is consistent with the constraint $h_b - h_a < 2 \wedge h_d - h_c > 2$ and then the state p_8 is declared reachable from p_0.

2.7 Diagonal Constraints Can Be Safely Removed

We will show that simple diagonal constraints can be removed from timed processes without reducing their behavioral properties. The proof of this result follows ideas in [BDGP98] and uses induction on the number of constraints to be removed. It consists of keeping the information on the truth of diagonal constraints in the states of the timed processes. Such a transformation induces an exponential blowup in the size of the initial timed process [BC05]. We will first mark states of the semantics of timed processes with a tuple of boolean values representing the truth of simple diagonal constraints with respect to the valuation in the states.

Assume that $\mathcal{P}$ is a timed process and the unique simple diagonal constraint occurring in $\mathcal{P}$ is the simple constraint $C = (h - h' \bowtie c)$, with $h, h' \in \mathcal{H}_\Sigma$, $\bowtie \in \{\leq, <, \geq, >\}$. Then to each state (p, v) of $[\![\mathcal{P}]\!]$, we assign the truth value of $v \in [\![C]\!]$. The resulting transition system, that we call the *marked semantics of* $\mathcal{P}$ does not modify the behavioral properties of $\mathcal{P}$.

Observation 74.1 The marked semantics of $\mathcal{P}$ and the semantics of $\mathcal{P}$ are isomorphic.

Now, from $\mathcal{P}$ and C, we build a new timed process $\mathcal{R}(\mathcal{P}, C) = \langle P \times \{0,1\}, \Sigma, (p^0, tv^0)), \Delta_\mathcal{R} \rangle$ where,

- tv^0 is the truth value of $v^0 \in [\![C]\!]$. We recall that v^0 is the valuation that assigns the constant 0 to every clock in $\mathcal{H}_\Sigma$
- $\Delta_\mathcal{R}$ is a transition relation defined according to Δ_P as follows: Let (p, tv) be a state of $\mathcal{R}$ and let $p \xrightarrow{g,a} p'$ be a transition in Δ_P. Then there are three cases for defining transition from (p, tv) depending on $p \xrightarrow{g,a} p'$ and C:
  1. if $h \neq h_a$ and $h' \neq h_a$, then we add the transition $(p, tv) \xrightarrow{g,a} (p', tv)$ in $\Delta_\mathcal{R}$.
  2. if $h' = h_a$ then the following transitions are added in $\Delta_\mathcal{R}$

(a) $(p,tv) \stackrel{g\wedge h\bowtie c,a}{\longrightarrow} (p',1)$

(b) $(p,tv) \stackrel{g\wedge\neg(h\bowtie c),a}{\longrightarrow} (p',0)$

3. The last case when $h = h_a$ is dual to the case just above.

To end the construction of $\mathcal{R}(\mathcal{P},C)$, we remove in it every transition of the form $(p,0) \stackrel{g,a}{\longrightarrow} (p',tv)$ where C occurs in g and we delete every occurrence of C in the remaining transitions.

We remark that by construction $\mathcal{R}(\mathcal{P},C)$ does not contain any occurrence of the simple diagonal constraint C and we show that $[\![\mathcal{R}(\mathcal{P},C)]\!]$ is bisimilar to the marked semantics of $\mathcal{P}$.

Lemma 75 Let $\mathcal{P}$ be a timed process and C be a simple diagonal constraint occurring in $\mathcal{P}$. The marked semantics of $\mathcal{P}$ and $[\![\mathcal{R}(\mathcal{P},C)]\!]$ are bisimilar.

Proof
Consider the relation R defined by $(p,v)R((p,tv),v)$ if tv is the truth value of $v \in [\![C]\!]$. We show that R is a bisimulation. As the cases of delay transitions are obvious we consider the cases of discrete transitions. Assume that $(p,v)R((p,tv),v)$.

1. if $tv = 0$ then $v \notin [\![C]\!]$.

   $\Longrightarrow$ Now assume that $(p,v) \stackrel{a}{\longrightarrow} (p',v')$, then $v' = v[h_a := 0]$ and there is $p \stackrel{g,a}{\longrightarrow} p'$ such that $v \in [\![g]\!]$ and C does not occurs in g as $v \notin [\![C]\!]$.

   (a) if $h \neq h_a$ and $h' \neq h_a$, then the transition $(p,0) \stackrel{g,a}{\longrightarrow} (p',0)$ exist in $\Delta_{\mathcal{R}}$ and because $v \in [\![g]\!]$ we get that $((p,0),v) \stackrel{a}{\longrightarrow} ((p',0),v')$. As neither h nor h' have been modified, $v' \notin [\![C]\!]$ and then $(p',v')R((p',0),v')$.

   (b) if $h' = h_a$ then, the transitions $(p,0) \stackrel{g\wedge h\bowtie c,a}{\longrightarrow} (p',1)$ and $(p,0) \stackrel{g\wedge\neg(h\bowtie c),a}{\longrightarrow} (p',0)$ exits in $\Delta_{\mathcal{R}}$. But either $v \in [\![h \bowtie c]\!]$ or $v \in [\![\neg(h \bowtie c)]\!]$.

      i. If $v \in [\![h \bowtie c]\!]$ then $v' \in [\![h - h_a \bowtie c]\!]$ as the clock h_a is reset after the transitions. Then we get that $((p,0),v) \stackrel{a}{\longrightarrow} ((p',1),v')$ and $(p',v')R((p',1),v')$

      ii. If $v \in [\![\neg(h \bowtie c)]\!]$ then $v' \notin [\![h - h_a \bowtie c]\!]$ as the clock h_a is reset after the transitions. Then we get that $((p,0),v) \stackrel{a}{\longrightarrow} ((p',0),v')$ and $(p',v')R((p',0),v')$

   (c) the last case when $h = h_a$ is dual to the case just above.

   $\Longleftarrow$ Now assume that $((p,0),v) \stackrel{a}{\longrightarrow} ((p',tv'),v')$. Then, there is $(p,0) \stackrel{g,a}{\longrightarrow} (p',tv')$ such that $v \in [\![g]\!]$ and C does not occur in g as $tv = 0$.

   (a) if $h \neq h_a$ and $h' \neq h_a$, then $tv' = tv = 0$ and $v' \notin [\![h - h' \bowtie c]\!]$ and there is a transition $p \stackrel{g,a}{\longrightarrow} p'$ in Δ_P. As $v \in [\![g]\!]$ we get that $(p,v) \stackrel{a}{\longrightarrow} (p',v')$ and $(p',v')R((p',0),v')$.

   (b) if $h' = h_a$ then $g = g' \wedge g''$ where $g'' = h \bowtie c$ or $g'' = \neg(h \bowtie c)$ and there is a transition $p \stackrel{g',a}{\longrightarrow} p'$ in Δ_P. As $v \in [\![g]\!]$ we get that $(p,v) \stackrel{a}{\longrightarrow} (p',v')$ and $(p',v')R((p',tv'),v')$ where $tv' = 1$ if $g'' = h \bowtie c$ otherwise $tv' = 0$.

   (c) the last case when $h = h_a$ is dual to the case just above.

2. The case when $tv = 1$ is similar to the case when $tv = 0$.

□

Let $\mathcal{P}$ be a timed process, and let $C_1, C_2, \ldots, C_n$ be all the simple diagonal constraints in $\mathcal{P}$. From the lemma above we can recursively remove each constraint obtaining a timed process without a diagonal constraint that preserves the behavioral properties. We get the following proposition.

Proposition 76 For every timed process $\mathcal{P}$ that uses diagonal constraints in its transition relation, there is a timed process $\mathcal{P}'$ that does not uses diagonal constraints in its transition relation such that $[\![\mathcal{P}]\!]$ is bisimilar to $[\![\mathcal{P}']\!]$.

2.8 Concluding Remarks

We have presented timed processes as models for real-time systems that use the time information on the occurrences of events for executing correctly.

We have recalled that regions provide a good abstraction for theoretical analysis of timed processes using diagonal free or general constraints. We have also presented the reachability analysis through zone-based abstraction and we have shown that when timed processes are defined using general constraints the zone-based approach combined with the approximation operator that we have considered leads to incorrect results. Then, we wondered if we could discard diagonal constraints from timed process without reducing their expressive power. The answer is yes and we have shown how to transform (with exponential cost) timed processes with diagonal constraint into equivalent behavioral timed processes with diagonal-free constraints only. As we will not be interested in efficient procedures, but in understanding models and their properties, in the following we will consider timed processes with diagonal-free constraints only.

Chapter 3

Results on Event-Recording Logic

The design of models for systems, and real-time systems in particular, is carried by requirements. Requirements describe desired or undesired properties of systems encompassing behavioral properties such as reachability, liveliness, deadlock and safety properties. For real-time systems, requirements must consider timing information. For example, a requirement must not only define the logical moment at which events (tasks) must occur (terminate), but must also describe the quantitative time information on occurrences of events (termination of tasks). Given a requirement, it is useful to check whether a given real-time system meets that requirement (model-checking), or to check whether we can construct a real-time system that meets the requirement (satisfiability checking).

We consider the logic Event-Recording Logic(ERL) as a formal language for describing properties on timed processes. Event-Recording Logic has been introduced by Sorea [Sor02] as a timed extension of the μ-calculus [Koz82]. In this logic, modalities are indexed both with an event and a constraint. We consider the basic problems about this logic such as the model-checking problem, the satisfiability checking problem and the equivalence between formulas and formulas in disjunctive normal form.

To solve the model-checking problem, we transform formulas into equivalent *rectangular formulas*. Rectangular formulas use only rectangular constraints. The later formulas are used by the model-checking procedure. We show that checking if a timed process is a model of a formula is equivalent to check if the M-region representation of that timed process is a model of the corresponding rectangular formula. Then, our model-checking procedure works as a model-checking procedure of the standard μ-calculus. Intuitively, in that procedure, for checking if $\varphi \vee \psi$ is satisfied in some state, it is enough to check if φ is satisfied in that state or ψ is satisfied in that state; such a step is a *non deterministic step*. For checking if a fixpoint formula $\sigma X.\varphi(X)$ is satisfied in some state, we check if $\varphi(\sigma X.\varphi(X))$ (regeneration step) is satisfied in that state. For checking is a conjunct $\varphi \wedge \psi$ is satisfied in some state, we check whether φ is satisfied in that state and whether ψ is also satisfied in that state. For checking if $\langle g, a\rangle\varphi$ is satisfied in some state, we check that there is an outgoing transition from that state labelled with (g, a) that leads to a state satisfying φ. For checking if $[g, a]\varphi$ is satisfied in some state, we check that every outgoing transition from that state labelled with the couple (g, a) leads to some state that satisfies φ.

From the intuitive idea for the model-checking procedure of ERL, we provide a tableau-based decision procedure for the satisfiability checking problem of ERL formulas. For that purpose, we define a *tableau system of rules* each rule of which is made of a conclusion and premise. Conclusions and premises are set of *timed sequents*. Each timed sequent is a tuple made of a set of a formulas and a timing context represented by a region. The use of a set of formulas in timed sequent is a consequence of the fact that to check if a formula of the $\varphi \wedge \psi$ is satisfied in some state, we must check that the state satisfies φ and ψ. We define the notion of a *tableau* and we show that a formula is satisfiable if and only if it contains a particular "good" sub-tableau. The "goodness" of a sub tableau is defined as the "goodness" of all the paths it contains and the "goodness" of a path is defined according to *t*races and the number of times fixpoint formulas are regenerated. Traces are links between formulas in premises and formulas in conclusions; they are useful as they also keep track of the regeneration of a fixpoint formula. As timed sequent may contain many fixpoint formulas, *signature* (tuple of ordinals) will be combined with traces to keep track of the number of time each fixpoint formula will be regenerated along a path. We also compare our satisfiability decision procedure with an earlier one proposed by Sorea [Sor02]. We get that the tableau system of rules of Sorea is somehow ambiguous as a particular rule may have two interpretations. Moreover the system of Sorea is incorrect due to the use of zones for representing timing context in the tableau for formulas that use diagonal constraints.

As timed sequents are labelled with set of formulas and we need to decompose a path into traces to decide if they are "good", the satisfiability procedure is expensive (exponential on the size of the formula) and difficult. Then we wondered if all these artifacts can be avoided and the answer is yes. An intuitive idea is to consider only conjunctions that do not require the use of sets of formulas in timed sequents. An example for such a conjunction is a conjunction of the form $\langle g_1, a_1\rangle\varphi \wedge \langle g_2, a_2\rangle\psi$ where $g_1 \wedge g_2$ is inconsistent or $a_1 \neq a_2$. Another idea consists, given a formula $\langle g_1, a\rangle\varphi \wedge \langle g_2, a\rangle\psi$ where $g_1 \wedge g_2$ is consistent, to assume that the models will have two outgoing transitions, one for each member of the conjunction. Then, we introduce the notion of *disjunctive normal form* for formulas. We show that every formula has an equivalent disjunctive normal form formula. From a given general ERL formula, we construct an equivalent ERL formula in disjunctive normal form. The satisfiability checking for disjunctive normal form formulas is easier than the satisfiability procedure of general formulas.

Related results: The standard (untimed) μ-calculus has been introduced by Kozen [Koz82]. Model-checking and satisfiability of the μ-calculus have been shown to have efficient (tableau-based) procedures [SE89, Eme97, GV08] and to have relations with other theoretical objects such as game [EJ91, NW96] and automata [Tho90, JW95]. For the later relation, a disjunctive normal form of formulas has been provided [JW95] to show the equivalence between alternating automata on trees and non-deterministic automata on trees [MS95]. The use of such kind of transformation has been presented in [AVW03, AW07, BCL05] for controller synthesis of systems. As ERL extends the μ-calculus, we have wondered if some of the results on the μ-calculus could be extended to ERL. Sorea [Sor02] has considered the tableau technique, early used in the setting of the (untimed) μ-calculus [JW95], to prove the decidability of the model-checking and satisfiability problems on ERL. The difficulty with the procedure of Sorea is that it is based on zones and rule that reduces modalities indexed with a constraint and an event is not easy to understand. We have proposed new rules and we have tried to reuse as much as possible standard results on the μ-calculus.

In the next section, we present ERL and its semantics. In Section 3.2, we consider the

model-checking problem of ERL formulas that we reduce to the model-checking problem of μ-calculus formulas modalities of which are indexed in a particular alphabet. The satisfiability problem for ERL formulas is addressed in Section 3.3. In that section we provide a new tableau system that uses regions for representing the timing context. From our tableau system we provide disjunctive normal form theorem in Section 3.5.

3.1 Event-Recording Logic

3.1.1 Definitions

Event-Recording Logic [Sor02] is an extension of the μ-calculus; it has been introduced to describe properties on timed processes. The extension is made on modal operators by considering modal operators of the form $\langle g, a\rangle$ and $[g, a]$.

Definition 77 Let Σ be a set of events, *Var* a set of variables. The set of formulas of Event-Recording Logic (ERL) denoted by $\mathcal{F}_{erl}$ is the set of formulas given by the following grammar:

$$\varphi ::= tt \mid ff \mid X \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle g, a\rangle\varphi \mid [g, a]\varphi \mid \mu X.\varphi \mid \nu X.\varphi$$

where,

- a is an event from Σ,
- g is a constraint from Gds_Σ,
- X is a variable from *Var*,

The *bound* of a formula is the maximal constant that occurs in its constraints. For a formula φ, M_φ denotes its bound. Given a constant M, we say that a formula is M-bounded if its bound is smaller than M. The notions of sub formula, free variable, binding , dependency order , expansion, sentence, guarded formula , positive formula for the setting of ERL are obvious from the definitions of similar notions for the setting of the μ-calculus in Section 1.3.

3.1.2 Semantics

Our goal is to interpret a formula φ of $\mathcal{F}_{erl}$ over timed processes. Because the meaning of a timed process is a $(\mathcal{V}_\Sigma \cup \Sigma)$-labelled transition system, we give the interpretation of a formula over such type of transition systems.

Notation: We will write $s \xrightarrow{v,a} s'$ when there is s'' such that $s \xrightarrow{v} s''$ and $s'' \xrightarrow{a} s'$.

As a formula may contain free variables we will need a valuation of such variables. Given a valuation of variables $Val : Var \to \mathcal{P}(S)$ and a set of states $T \subseteq S$, the valuation $Val[X/T]$ is the valuation *Val* with the substitution that associates the set of states T with the variable X. Formally, for $Y \in Var$, $Val[X/T](Y) = T$ if $Y = X$ and $Val(Y)$ otherwise. We write $\mathcal{S}, s, Val \vDash_t \varphi$ when the formula φ holds in s or equivalently s satisfies φ.

Definition 78 (Meaning of a formula over $(\mathcal{V}_\Sigma \cup \Sigma)$-labelled transition systems) For a given $(\mathcal{V}_\Sigma \cup \Sigma)$-labelled transition system $\mathcal{S}$, a given formula φ and an assignment $Val : Var \to \mathcal{P}(S)$, we define the satisfaction relation $\models_t$ and the set $[\![\varphi]\!]^{\mathcal{S}}_{Val}$ inductively as follows:

- $\mathcal{S}, s, Val \models_t tt$.
- $\mathcal{S}, s, Val \models_t X$ if $s \in Val(X)$.
- $\mathcal{S}, s, Val \models_t \varphi_1 \vee \varphi_2$ if $\mathcal{S}, s, Val \models_t \varphi_1$ or $\mathcal{S}, s, Val \models_t \varphi_2$.
- $\mathcal{S}, s, Val \models_t \varphi_1 \wedge \varphi_2$ if $\mathcal{S}, s, Val \models_t \varphi_1$ and $\mathcal{S}, s, Val \models_t \varphi_2$.
- $\mathcal{S}, s, Val \models_t [g, a]\psi$ if for every $s \xrightarrow{v,a} s' \in \Delta_{\mathcal{S}}$ such that $v \in [\![g]\!]$ we have $\mathcal{S}, s', Val \models_t \psi$.
- $\mathcal{S}, s, Val \models_t \langle g, a\rangle\psi$ if there exists $s \xrightarrow{v,a} s' \in \Delta_{\mathcal{S}}$ such that $v \in [\![g]\!]$ and $\mathcal{S}, s', Val \models_t \psi$.
- $\mathcal{S}, s, Val \models_t \mu X.\varphi(X)$ if $s \in \bigcap\{T \subseteq S \mid [\![\varphi(X)]\!]^{\mathcal{S}}_{Val[X/T]} \subseteq T\}$.
- $\mathcal{S}, s, Val \models_t \nu X.\varphi(X)$ if $s \in \bigcup\{T \subseteq S \mid T \subseteq [\![\varphi(X)]\!]^{\mathcal{S}}_{Val[X/T]}\}$.
- $[\![\varphi]\!]^{\mathcal{S}}_{Val} = \{s \mid \mathcal{S}, s, Val \models_t \varphi\}$.

We will sometimes write $s \in [\![\varphi]\!]^{\mathcal{S}}_{Val}$ instead of $\mathcal{S}, s, Val \models_t \varphi$. If φ is a *sentence*, i.e., does not have free variables, then its meaning does not depend on a valuation and we can write just $\mathcal{S}, s \models_t \varphi$. Finally, we will write $\mathcal{S} \models_t \varphi$ for $\mathcal{S}, s^0 \models_t \varphi$ to say that $\mathcal{S}$ is a *model* of φ.

Let us consider φ a formula and $\mathcal{P}$ a timed process. We say that φ is satisfied in a state p, a valuation $v : H \to \mathbb{R}^+$ and a valuation $Val : Var \to \mathcal{P}(P \times \mathcal{V}_\Sigma)$ of propositional variables and we write $\mathcal{P}, (p, v), Val \models \varphi$ when $[\![\mathcal{P}]\!], (p, v), Val \models_t \varphi$.

Definition 79 (Meaning of a formula over timed processes) The meaning $[\![\varphi]\!]^{\mathcal{P}}_{Val} \subseteq P \times (\mathcal{H} \to \mathbb{R}^+)$ of a formula over a timed process $\mathcal{P}$ is defined by :

$$[\![\varphi]\!]^{\mathcal{P}}_{Val} = [\![\varphi]\!]^{[\![\mathcal{P}]\!]}_{Val}$$

We will write $\mathcal{P} \models \varphi$ if $[\![\mathcal{P}]\!]$ is a model of φ and we say that $\mathcal{P}$ is a *model* of φ.

Given two formulas φ_1 and φ_2, we often use the notation $\varphi_1 \equiv \varphi_2$ to say that φ_1 is *equivalent* to φ_2, meaning that for every timed process $\mathcal{P}$, $[\![\varphi_1]\!]^{\mathcal{P}}_{Val} = [\![\varphi_2]\!]^{\mathcal{P}}_{Val}$.

Remark: The presentation of the semantics above is different (but it is equivalent) from the one in [Sor02]. In particular, the presentation of the semantics of modal operators indexed with a constraint and an event seems simpler as it benefits from that delay transitions in the semantics of timed processes (see Definition 66) are labelled with valuations.

Given the sentence φ, and a $(\mathcal{V} \cup \Sigma)$-labelled transition system $\mathcal{S}$, we introduce the negation operator $\neg$ defined by: $[\![\neg\varphi]\!]^{\mathcal{S}} = S \setminus [\![\varphi]\!]^{\mathcal{S}}$.

Proposition 80 The following equivalences are true:

- $\neg tt \equiv ff$

- $\neg \mathit{ff} \equiv tt$
- $\neg(\varphi_1 \wedge \varphi_2) \equiv \neg\varphi_1 \vee \neg\varphi_2$
- $\neg(\varphi_1 \vee \varphi_2) \equiv \neg\varphi_1 \wedge \neg\varphi_2$
- $\neg\langle g, a\rangle\varphi \equiv [g, a]\neg\varphi$
- $\neg[g, a]\varphi \equiv \langle g, a\rangle\neg\varphi$
- $\neg\mu X.\varphi(X) \equiv \nu X.\neg\varphi(\neg X)$
- $\neg\nu X.\varphi(X) \equiv \mu X.\neg\varphi(\neg X)$

Proof
Let a sentence φ, a $(\mathcal{V} \cup \Sigma)$-labelled transition system $\mathcal{S}$ and let s be a state of $\mathcal{S}$. The proof uses structural induction. All the cases but for modalities are standard.

- If $s \in [\![\neg\langle g, a\rangle\varphi]\!]^{\mathcal{S}}$ then $s \notin [\![\langle g, a\rangle\varphi]\!]^{\mathcal{S}}_{Val}$. By definition it means that for every $v' \in [\![g]\!]$, $s \xrightarrow{v',a} s'$ in $\mathcal{S}$ we have $s' \notin [\![\varphi]\!]^{\mathcal{S}}_{Val}$. Once again by definition $s \in [\![[g, a]\neg\varphi]\!]^{\mathcal{S}}_{Val}$.
- The case of $\neg[g, a]\varphi$ uses a dual argumentation.

□

We can assume that the grammar of the syntax of ERL is augmented with the negation operator in the following way: if φ is an ERL sentence, so is $\neg(\varphi)$.

Proposition 81 Let $g, g_1, g_2, \ldots, g_n \in \mathit{Gds}_\Sigma$ be such that $[\![g]\!] = \bigcup_{i=1..n} [\![g_i]\!]$ then,

1. $\langle g, a\rangle\varphi \equiv \bigvee_{i=1..n}\langle g_i, a\rangle\varphi$
2. $[g, a]\varphi \equiv \bigwedge_{i=1..n}[g_i, a]\varphi$

Proof
Let $\mathcal{S}$, s a configuration of s, and Val a valuation of propositional variables.

1. $\mathcal{S}, s, \mathit{Val} \vDash_t \langle g, a\rangle\varphi$ if and only if there is $v \in [\![g]\!]$ such that $s \xrightarrow{v,a} s'$ and $\mathcal{S}, s', \mathit{Val} \vDash_t \varphi$. Equivalently, $\mathcal{S}, s, \mathit{Val} \vDash_t \langle g_i, a\rangle\varphi$ for some $i \in \{1, 2, \ldots, n\}$ such that $v \in g_i$ and equivalently $\mathcal{S}, s, \mathit{Val} \vDash_t \bigvee_{i=1,\ldots,n}\langle g_i, a\rangle\varphi$.
2. From proposition 80 $[\![[g, a]\varphi]\!] = [\![\neg(\langle g, a\rangle\neg\varphi)]\!]$. Then, use the first item to conclude.

□

3.2 Model-Checking

In this section we address the model-checking problem that is to check if a given a system is a model of a given specification. In the literature (see [CGP99, Mer01, BBF+01, GV08]), this problem is known as the model-checking problem and it has been widely studied for several temporal logics [Var96, LP85, KVW00, Sch03, Ong02, LMS04] and particularly the μ-calculus [SE89, NW96, Eme97].

Here, we present an algorithm for the model-checking problem where, the system, $\mathcal{P}$ is a timed process and the specification, φ is a formula of ERL. That algorithm is based on the one for the μ-calculus. Indeed, we show that checking if a timed process is a model of an ERL formula can be reduced to checking whether an untimed transition system is a model of a particular μ-calculus formula.

Let $\mathcal{S} = \langle S, \mathcal{V}_\Sigma \cup \Sigma, s, \Delta_\mathcal{S} \rangle$ be a $(\mathcal{V}_\Sigma \cup \Sigma)$-labelled transition system, φ an ERL formula. Informally, to check whether $\mathcal{S}$ is a model of φ, we work in the following way. First we assign the initial configuration s^0 to φ meaning that we are checking if $s^0 \models_t \varphi$. Now, assume that a configuration s has been assigned to a sub formula ψ of φ. Depending of the the structure of ψ, we must check if some (or all) successors of s in $\mathcal{S}$ satisfy some sub formula of ψ.

When φ is a sentence and does not contain free variables, the procedure works as follows:

- if $\psi = tt$ the answer if "yes";
- if $\psi = ff$ the answer is "no";
- to verify that $\varphi_1 \wedge \varphi_2$ is satisfied at s, we check that φ_1 is satisfied at s and φ_2 is satisfied at s;
- to verify that $\varphi_1 \vee \varphi_2$ is satisfied at s, we check if φ_1 is satisfied at s or if φ_2 is satisfied at s;
- to verify that $\langle g, a \rangle \psi$ is satisfied at s, we check that s' satisfies ψ where s' is a chosen configuration such that $s \xrightarrow{v',a} s'$ in $\mathcal{S}$ and $v' \in [\![g]\!]$;
- to verify that $[g, a]\psi$ is satisfied at s, we check that s' satisfies ψ for every s' such that $s \xrightarrow{v',a} s'$ in $\mathcal{S}$ and $v' \in [\![g]\!]$;
- to verify that $\sigma X.\psi(X)$ is satisfied at s, we check $\psi(X)$ is satisfied at s;
- to verify that X is satisfied at s, we check that $\psi(X)$ is satisfied at s; assuming that X is bound and $\mathcal{D}_\varphi(X) = \sigma X.\psi(X)$.

When φ is not a sentence, we need a valuation of free variables over the set of states S.

To check whether a timed process $\mathcal{P}$ is a model of a formula φ, means to check whether $[\![\mathcal{P}]\!]$ is a model of φ. One could wonder about the termination of such a procedure since the state space of $[\![\mathcal{P}]\!]$ is infinite and there could exists infinitely many outgoing transitions from configurations of $[\![\mathcal{P}]\!]$.

We intend to operate on a finite structure and try to use decision procedures for the model-checking problem of the μ-calculus. We consider abstraction on models that preserve their semantics.

1. We consider "good abstractions" for timed processes. These abstractions will be finite labelled transition systems.

2. We consider "good abstraction" for formulas. These abstractions will be formulas of finite length.

3. We define a "good abstract" satisfaction relation between a "good abstraction" of a timed process and "good abstraction" of a formula. This "good abstract" satisfaction relation must be defined in such a way that a timed process satisfies a formula if and only if the abstraction of the timed process satisfies the abstraction of the formula.

3.2.1 Abstract Semantics for Formulas

We propose in Definition 82 the symbolic relation, denoted by $\models_g$, of satisfaction between a timed process and a formula. This relation will serve as a "good abstract" satisfaction relation we have discussed before. We immediately remark that for any timed process $\mathcal{P}$: $(\!|\mathcal{P}|\!)^M$ and $(\!|\mathcal{P}|\!)^M_{reg}$ are $(Agds_\Sigma(M) \cup \Sigma)$-LTS. $(\!|\mathcal{P}|\!)^M$ has infinitely many states while $(\!|\mathcal{P}|\!)^M_{reg}$ has finitely many states. Our objective will be to reduce the model checking over $[\![\mathcal{P}]\!]$ to the model checking over $(\!|\mathcal{P}|\!)^M_{reg}$ for suitable M.

Notation: We will write $s \xrightarrow{g,a} s'$ when there is s'' such that $s \xrightarrow{g} s''$ and $s'' \xrightarrow{a} s'$.

Definition 82 [Abstract meaning of a formula over a $(Gds_\Sigma \cup \Sigma)$-LTS)] Let φ be a formula, $\mathcal{S} = \langle S, Gds_\Sigma \cup \Sigma, s^0, \Delta_\mathcal{S}\rangle$ be a $(Gds_\Sigma \cup \Sigma)$-LTS. For a configuration s of $\mathcal{S}$, a valuation $Val : Var \to \mathcal{P}(S)$ of propositional variables, we define the *symbolic relation of satisfaction* $\models_g$ and the set ${}^g[\![\varphi]\!]^\mathcal{S}_{Val}$ as follows:

- $\mathcal{S}, s, Val \models_g tt$.
- $\mathcal{S}, s, Val \models_g X$ if $s \in Val(X)$.
- $\mathcal{S}, s, Val \models_g \varphi_1 \vee \varphi_2$ if $\mathcal{S}, s, Val \models_g \varphi_1$ or $\mathcal{S}, s, Val \models_g \varphi_2$.
- $\mathcal{S}, s, Val \models_g \varphi_1 \wedge \varphi_2$ if $\mathcal{S}, s, Val \models_g \varphi_1$ and $\mathcal{S}, s, Val \models_g \varphi_2$.
- $\mathcal{S}, s, Val \models_g [g, a]\psi$ if for every $s \xrightarrow{g,a} s' \in \Delta_\mathcal{S}$, we have $\mathcal{S}, s', Val \models_g \psi$.
- $\mathcal{S}, s, Val \models_g \langle g, a\rangle\psi$ if there exist $s \xrightarrow{g,a} s' \in \Delta_\mathcal{S}$ and $\mathcal{S}, s', Val \models_g \psi$.
- $\mathcal{S}, s, Val \models_g \mu X.\varphi(X)$ if $s \in \bigcap\{T \subseteq S \mid {}^g[\![\varphi(X)]\!]^\mathcal{S}_{Val[X/T]} \subseteq T\}$.
- $\mathcal{S}, s, Val \models_g \nu X.\varphi(X)$ if $s \in \bigcup\{T \subseteq S \mid T \subseteq^g [\![\varphi(X)]\!]^\mathcal{S}_{Val[X/T]}\}$.
- ${}^g[\![\varphi]\!]^\mathcal{S}_{Val} = \{s : \mathcal{S}, s, Val \models_g \varphi\}$

If φ is a *sentence*, i.e., does not have free variables, then its meaning does not depend on a valuation and we can write just $\mathcal{S}, s \models_g \varphi$. Finally, we will write $\mathcal{S} \models_g \varphi$ for $\mathcal{S}, s^0 \models_g \varphi$.

We propose the symbolic decision procedure for the model checking of a process $\mathcal{P}$ against a formula ϕ. This procedure is similar to the real-time decision procedure defined above. The difference occurs when we check if a state satisfies a sub formula of the form $\langle g, a\rangle\psi$ and $[g, a]\psi$. In these cases, the procedure works as follows:

- to verify that $\langle g, a\rangle\psi$ is satisfied at s, we check that s' satisfies ψ where s' is a chosen configuration such that $s \xrightarrow{g,a} s'$ in $\mathcal{S}$
- to verify that $[g, a]\psi$ is satisfied at s, we check that s' satisfies ψ for every s' such that $s \xrightarrow{g,a} s'$ in $\mathcal{S}$

It should be quite clear that for timed processes, we can not just substitute the symbolic decision procedure for the real-time decision procedure.

We expect to apply the symbolic decision procedure on representations of timed processes and formulas that use constraints in a same finite set. Then, we will consider bounded constraints, and we will ensure that any constraint in that set can not be split into constraints that use smaller constants. According to Fact 48, rectangular constraints are appropriate for this objective.

Definition 83 An *M-rectangular formula* is a formula using constraints in $Agds_{\Sigma}(M)$.

Given a formula φ, and a bound M, we define the M-rectangular formula $Rect_M(\varphi)$ as the formula obtained from φ by replacing each constraint g that occurs in φ by the disjunction of atomic M-rectangular constraints contained in g. From Proposition 81, this definition is sound and $Rect_M(\varphi)$ is a formula of $\mathcal{F}_{erl}$.

Definition 84 The M-rectangular ERL formula associated to an ERL formula φ, $Rect_M(\varphi)$ is the formula defined inductively as follows:

- $Rect_M(ff) = ff$
- $Rect_M(tt) = tt$
- $Rect_M(X) = X$
- $Rect_M(\varphi_1 \wedge \varphi_2) = Rect_M(\varphi_1) \wedge Rect_M(\varphi_2)$
- $Rect_M(\varphi_1 \vee \varphi_2) = Rect_M(\varphi_1) \vee Rect_M(\varphi_2)$
- $Rect_M(\langle g, a\rangle\varphi) = \bigvee_{\hat{g}\in Rect_M(g)} \langle \hat{g}, a\rangle Rect_M(\varphi)$
- $Rect_M([g, a]\varphi) = \bigwedge_{\hat{g}\in Rect_M(g)} [\hat{g}, a] Rect_M(\varphi)$
- $Rect_M(\sigma X.\varphi(X)) = \sigma X.Rect_M(\varphi(X))$ where σ is one of $\{\mu, \nu\}$

We remark that the size of $Rect_M(\varphi)$ could be exponential in the size of φ.

We show in the following proposition that for $M \geq M_\varphi$ (M_φ is the maximal constant that occurs in φ) formulas and their M-rectangular forms are equivalent over transition systems that represents the semantics of timed processes.

Proposition 85 Let φ be a formula and $\mathcal{S}$ be a $(\mathcal{V}_\Sigma \cup \Sigma)$-LTS. For every $M \geq M_\varphi$, $\mathcal{S}, s, Val \models_t \varphi$ if and only if $\mathcal{S}, s, Val \models_t Rect_M(\varphi)$.

Proof
We use structural induction. Let $M \geq M_\varphi$.

- The basic cases of ff, tt, X are obvious.
- The cases of $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$ are also obvious.
- The case when $\varphi = \langle g, a \rangle \psi$. $Rect_M(\varphi) = \bigvee_{\hat{g} \in Rect_M(g)} \langle \hat{g}, a \rangle \varphi$. From Proposition 57, $[\![g]\!] = \bigvee_{\hat{g} \in Rect_M(g)} [\![\hat{g}]\!]$. We use Proposition 81 to conclude.
- The case when $\varphi = [g, a]\psi$. $Rect_M(\varphi) = \bigwedge_{\hat{g} \in Rect_M(g)} [\hat{g}, a] \varphi$. We use Proposition 57 and Proposition 81 to conclude.
- When $\varphi = \sigma X.\psi(X)$. $Rect_M(\varphi) = \sigma X.Rect_M(\psi(X))$. By induction hypothesis $\psi(X) \equiv Rect_M(\psi(X))$, then we get the result.

□

The "good abstraction" for formulas that we use later for the model-checking algorithm is the rectangular form.

3.2.2 Fixpoint Approximation

Now we introduce the notion of computation of a fixpoint by successive steps that gives us a powerful tool to understand the semantics of formulas. Let $\mathcal{S}$ be a $(Gds_\Sigma \cup \Sigma)$-LTS.

Definition 86 For every ordinal λ, we define constructions for the fixpoints $\mu^\lambda X.\varphi(X)$ and $\nu^\lambda X.\varphi(X)$ the semantics of which is inductively defined as follows:

- ${}^g[\![\mu^0 X.\varphi(X)]\!]^{\mathcal{S}}_{Val} = \emptyset$, and ${}^g[\![\nu^0 X.\varphi(X)]\!]^{\mathcal{S}}_{Val} = {}^g[\![tt]\!]^{\mathcal{S}}_{Val}$.
- ${}^g[\![\sigma^{\lambda+1} X.\varphi(X)]\!]^{\mathcal{S}}_{Val} = {}^g[\![\varphi(X)]\!]^{\mathcal{S}}_{Val[{}^g[\![\sigma^\lambda X.\varphi(X)]\!]^{\mathcal{S}}_{Val}/X]}$
- When β is a limit ordinal,
  - ${}^g[\![\mu^\beta X.\varphi(X)]\!]^{\mathcal{S}}_{Val} = \bigcup_{\lambda<\beta} {}^g[\![\mu^\lambda X.\varphi(X)]\!]^{\mathcal{S}}_{Val}$,
  - ${}^g[\![\nu^\beta X.\varphi(X)]\!]^{\mathcal{S}}_{Val} = \bigcap_{\lambda<\beta} {}^g[\![\nu^\lambda X.\varphi(X)]\!]^{\mathcal{S}}_{Val}$.

We recall the Knaster-Tarski theorem [Tar55] showing how to approximate fixpoint by iterative computations.

Theorem 87 ([Tar55])

- $\mathcal{S}, s, Val \models_g \mu X.\varphi(X)$ *if and only if* $s \in \bigcup_\beta {}^g[\![\mu^\beta X.\varphi(X)]\!]^{\mathcal{S}}_{Val}$.
- $\mathcal{S}, s, Val \models_g \nu X.\varphi(X)$ *if and only if* $s \in \bigcap_\beta {}^g[\![\nu^\beta X.\varphi(X)]\!]^{\mathcal{S}}_{Val}$.

A definition and a theorem analogous to Definition 86 and Theorem 87 is made considering $(\mathcal{V}_\Sigma \cup \Sigma)$-LTS, $\models_t$ and $[\![\varphi]\!]$ instead of $(Gds_\Sigma \cup \Sigma)$-LTS, $\models_g$ and ${}^g[\![\varphi]\!]$.

3.2.3 Model-Checking Results

The next step in our abstraction is to show that the real-time algorithm for the model-checking problem, defined above, can be replaced by a symbolic algorithm. In order words, we make a relation between $\models_t$ and $\models_g$. Given a timed process $\mathcal{P}$ and an M_φ-rectangular formula φ, we show that the result of the "real-time" decision procedure for $[\![\mathcal{P}]\!] \models_t \varphi$ is the same as the result of the symbolic decision procedure for $\langle\!|\mathcal{P}|\!\rangle^M \models_g \varphi$ with $M \geq M_\varphi$. Recall that in aforementioned real-time algorithm as in the symbolic algorithm, the verification task consists in moving from a verification goal into another and the verdict depends on the number of times particular variables are regenerated. If we show that a succession of moves in the real-time procedure can be mimicked in the symbolic procedure in such a way that formulas and the locations in the verification goal are preserved then the verdict of the two procedures will always be the same. That is what we show in Lemma 88 below.

Lemma 88 For every $M_{\mathcal{P}}$-bounded process $\mathcal{P}$, for every M_φ rectangular formula, for every $M \geq M_\varphi$, $[\![\mathcal{P}]\!], (p, v), \mathit{Val} \models_t \varphi$ if and only if $\langle\!|\mathcal{P}|\!\rangle^M, (p, v), \mathit{Val} \models_g \varphi$.

Proof
The proof uses structural induction. Let $M \geq M_\varphi$.

- The basics cases when φ is of the form tt, ff, X are obvious.
- The cases of $\varphi_1 \wedge \varphi_2$, or $\varphi_1 \vee \varphi_2$ are also obvious.
- Assume that $\varphi = \langle g, a\rangle\psi$,

  (=>) If $[\![\mathcal{P}]\!], (p, v), \mathit{Val} \models_t \langle g, a\rangle\psi$ then there is $v' \in [\![g]\!]$ such that $(p, v) \xrightarrow{v', a} (p', v'[h_a := 0])$, and $[\![\mathcal{P}]\!], (p', v'[h_a := 0]), \mathit{Val} \models_t \psi$. By induction hypothesis $\langle\!|\mathcal{P}|\!\rangle^M, (p', v'[h_a := 0]), \mathit{Val} \models_g \psi$. But, $v' \models g$ implies that $(p, v) \xrightarrow{g, a} (p', v'[h_a := 0])$ and then $\langle\!|\mathcal{P}|\!\rangle^M, (p, v), \mathit{Val} \models_g \langle g, a\rangle\psi$.

  (<=) If $\langle\!|\mathcal{P}|\!\rangle^M, (p, v), \mathit{Val} \models_g \langle g, a\rangle\psi$, then there is $v' \in [\![g]\!]$, $(p, v) \xrightarrow{g, a} (p', v'[h_a := 0])$ in $\langle\!|\mathcal{P}|\!\rangle^M$ such that $\langle\!|\mathcal{P}|\!\rangle^M, (p', v'[h_a := 0]) \models_g \psi$. By induction hypothesis $[\![\mathcal{P}]\!], (p', v'[h_a := 0]), \mathit{Val} \models_t \psi$. But, if $(p, v) \xrightarrow{g, a} (p', v'[h_a := 0])$ in $\langle\!|\mathcal{P}|\!\rangle^M$, then there is $(p, v) \xrightarrow{v', a} (p', v'[h_a := 0])$ in $[\![\mathcal{P}]\!]$. This implies that $[\![\mathcal{P}]\!], (p, v), \mathit{Val} \models_t \langle g, a\rangle\psi$.

- The case when $\varphi = [g, a]\psi$ uses a dual argument to the case when $\varphi = \langle g, a\rangle\psi$.
- Since intersection and union of set preserves monotonicity, the cases of fixpoint formula come as a consequence of the above ones

□

As $\langle\!|P|\!\rangle^M$ has infinitely many states, the real-time model-checking procedure may not terminate. By the following proposition, we can consider the M-region representation that is finite to ensure termination. The following corollary is just an extension of Theorem 26 to timed processed and ERL formulas.

Corollary 89 Let $\mathcal{S}$ and $\mathcal{S}'$ be two bisimilar timed processes and let φ be a formula. $\mathcal{S} \models_g \varphi$ if and only if $\mathcal{S} \models_g \varphi$

Proposition 90 For any timed process $\mathcal{P}$, any formula φ, for every $M \geq \max(M_\varphi, M_\mathcal{P})$, $[\![\mathcal{P}]\!], (p, v), \mathit{Val} \models_t \varphi$ if and only if $\langle\!|\mathcal{P}|\!\rangle^M_{reg}, (p, [v]_M), \mathit{Val} \models_g \mathit{Rect}_M(\varphi)$

Proof
By Proposition 85, $[\![\mathcal{P}]\!], (p, v), \mathit{Val} \models_t \varphi$ if and only if $[\![\mathcal{P}]\!], (p, v), \mathit{Val} \models_t \mathit{Rect}_{M'}(\varphi)$ for every $M' \geq M_\varphi$. From Lemma 88, this is equivalent to $\langle\!|\mathcal{P}|\!\rangle^M, (p, v), \mathit{Val} \models_g \mathit{Rect}_{M'}(\varphi)$ for every $M' \geq M_\varphi$. From Proposition 69, $\langle\!|\mathcal{P}|\!\rangle^M$ is bisimilar to $\langle\!|\mathcal{P}|\!\rangle^M_{reg}$ then using Corollary 89, we get $\langle\!|\mathcal{P}|\!\rangle^M, (p, v), \mathit{Val} \models_g \mathit{Rect}_{M'}(\varphi)$ if and only if $\langle\!|\mathcal{P}|\!\rangle^M_{reg}, (p, [v]_M), \mathit{Val} \models_g \mathit{Rect}_M(\varphi)$ □

We get the following theorem.

Theorem 91 *There is an effective procedure that checks whether a timed process $\mathcal{P}$ is an model of a formula φ assuming an initial valuation v^0.*

Proof
By Proposition 90, we get that to check if $\mathcal{P}$ satisfies φ is equivalent to check that $\langle\!|\mathcal{P}|\!\rangle^M_{reg} \models_g \mathit{Rect}_M(\varphi)$ for M sufficiently big. But the relation $\models_g$ between $\langle\!|\mathcal{P}|\!\rangle^M_{reg}$ and $\mathit{Rect}_M(\varphi)$ is the same (modulo the labels on transition of the models and the index in the modalities) as the relation $\models$ between a labelled transition system and a μ-calculus formula. Then it comes that the model-checking procedure for the μ-calculus can be used for the model-checking of ERL. □

3.2.4 Complexity

The complexity for our model-checking algorithm is immediate from the complexity of the model-checking problem for the μ-calculus. In Theorem 25, the complexity of the model-checking for the μ-calculus depends on the size of the models, the alternation depth and the size (number of sub formulas) of the formulas. For the model-checking algorithm of ERL, models are M-region representations and formulas are M-rectangular. Let M be an integer, there are at most $(2 \times M + 1)^{|\mathcal{H}_\Sigma|}$ rectangular constraints. For a timed process $\mathcal{P} = \langle P, \Sigma, p^0, \Delta_P \rangle$, the M-region region representation $\langle\!|\mathcal{P}|\!\rangle^M_{reg}$ has at most $|P| \times |\mathit{Reg}(M)|$ states and at most $|\Delta_P| \times |\mathit{Agds}(M)|$ transitions. The M-rectangular formula $\mathit{Rect}_M(\varphi)$ for a given formula φ has at most $|\mathit{sub}(\varphi)| \times |\mathit{Agds}(M)|$ sub formulas and is of the same alternation depth as φ.

The we get the following corollary.

Corollary 92 Let φ be a formula, and $\mathcal{P} = \langle P, \Sigma, p^0, \Delta_P \rangle$ be a timed process. Our model-checking problem procedure procedure works in

$$\mathcal{O}\left(|\Delta_P| \times |\mathit{Agds}(M)|^2 \times |\mathit{sub}(\varphi)| \times \left(\frac{2 \times |P| \times |\mathit{Reg}(M)| \times |\mathit{sub}(\varphi)| \times |\mathit{Agds}(M)|}{\mathit{alt}(\varphi)}\right)^{\lceil \mathit{alt}(\varphi)/2 \rceil}\right)$$

We recall that $\mathit{alt}(\varphi)$ is the alternation depth of the formula φ.

From corollary 92, we get that our model-checking algorithm is exponential time in the size of the timed process and the length of the binary encoding of the largest constant in the formula and in the timed process.

We remark that a standard zone-based model-checking algorithm for WT_μ will remain in EXPTIME; but it may be an interesting alternative in practice.

3.3 Satisfiability

The *satisfiability problem* for ERL is: given a sentence of Event-Recording Logic φ, check whether there exists a timed process $\mathcal{P}$ that is a model of φ with respect the initial valuation v^0. By Propositions 56 and 81 we can assume that the formula uses only rectangular constraints.

We consider this problem and we propose a tableau-based decision procedure. We propose a new tableau system of rules that we use to construct tableaux. From tableaux we check the existence of "good" fragments to decide if formulas are satisfiable. In our tableau system of rules, timing contexts are represented by regions while the tableau system of rules proposed by Sorea [Sor02] for the satisfiability of ERL uses zones. Later in Section 3.4 we make some observations on the interpretations of some rules of Sorea and we show that it is incorrect for the satisfiability of a formula that uses general (diagonal) constraints as originally considered by Sorea.

Let us first extend the syntax of Event-recording logic formula by considering the new modal operator $(g,a) \rightarrow$ that extends the operator $(a) \rightarrow$ for the μ-calculus and that has the same expressive power that the former ones that is $\langle g,a\rangle$ and $[g,a]$.

Definition 93 Let Γ be a set of formulas. We define the modal operator $(g,a) \rightarrow\Gamma$ by:

$$(g,a) \rightarrow\Gamma \equiv \bigwedge_{\varphi\in\Gamma} \langle g,a\rangle\varphi \wedge [g,a] \bigvee_{\varphi\in\Gamma} \varphi$$

Recall that the conjunction of an empty set is equal to tt and the disjunction of an empty set is equal to ff; So $(g,a) \rightarrow\emptyset \equiv [a,a]ff$.

We immediately remark that:

$$\begin{aligned} [g,a]\varphi &\equiv (g,a) \rightarrow\{\varphi\} \vee (g,a) \rightarrow\emptyset \\ \langle g,a\rangle\varphi &\equiv (g,a) \rightarrow\{\varphi, tt\} \end{aligned}$$

In consequence, every formula of event-recording logic is equivalent to a formula using only the new modal operator. Later we consider rectangular formulas that only use the new modality operator in addition to the boolean and fixpoints operators.

3.3.1 Tableau

As checking if $\mathcal{P}$ is a model of φ consists to check if $[\![\mathcal{P}]\!] \vDash_t \varphi$, we describe the set of formulas, that we called the *satisfiability objective*, assigned to configurations (or state) of a $(\mathcal{V}\cup\Sigma)$-LTS that models a formula φ. This assignment procedure will give intuitions for the rules of the

tableau. Assignments map satisfiability objectives to states from which there is at least one transition labelled with a valuation.

Initially, $\{\varphi\}$ is assigned to the initial state s^0. The tuple (s, Γ) of a state s and its *satisfiability objective* Γ is a *verification objective*. We write $\{\varphi, \Gamma\}$ as a shortcut for $\{\varphi\} \cup \Gamma$.

Assuming that the current verification objective is $(s, \{\varphi, \Gamma\})$, and depending on the structure of φ, we use the verification procedure described in the previous section to generate the next satisfiability objectives. We first consider the cases when only the satisfiability objective changes. According to the verification procedure the change is done by the following rules:

- if $\varphi = \varphi_1 \wedge \varphi_2$, then the next satisfiability objective is $\{\varphi_1, \varphi_2, \Gamma\}$ since we would like to verify that s satisfies φ_1 and s satisfies φ_2;
- if $\varphi = \varphi_1 \vee \varphi_2$, then the next satisfiability objective is $\{\varphi_1, \Gamma\}$ or $\{\varphi_2, \Gamma\}$ since we would like to verify that s satisfies φ_1 or s satisfies φ_2;
- if $\sigma X.\psi(X)$, then the next satisfiability objective is $\{\psi(\sigma X.\psi(X)), \Gamma\}$. This is a *regeneration* step.

From a verification objective (s, Γ), if we apply the rules above, we do not change the current state in the transition system, but, we end up in a verification objective (s, Γ) where Γ is such that every formula in it is in one of the form tt, ff, or $(g, a) \rightarrow \Theta$. Then, we consider the following cases:

- if Γ contains the formula ff, then Γ is not satisfied in s;
- if tt is the unique formula in then Γ is satisfied in s;
- else, Γ contains at least one formula of the form $(g, a) \rightarrow \Theta$ then:
  - for every $(g, a) \rightarrow \Theta \in \Gamma$, and $\psi \in \Theta$, we must create a verification objective $(s', \{\psi\} \cup \{\bigvee \Theta' \mid (g', a) \rightarrow \Theta' \in \Gamma \text{ and } v \in [\![g']\!]\})$ for some s' and $v \in [\![g]\!]$ with $s \xrightarrow{v,a} s'$,
  - for every (v, a) and every s' such that $s \xrightarrow{v,a} s'$, we must find a formula $(g', a) \rightarrow \Theta \in \Gamma$ with $v \in [\![g']\!]$ and $\psi \in \Theta$; and we create the verification objective $(s', \{\psi\} \cup \{\bigvee \Theta' \mid (g'', a) \rightarrow \Theta' \in \Gamma \text{ and } v \in [\![g']\!]\})$.

As a consequence of Lemma 88, considering $[\![\mathcal{P}]\!]$ and $\langle\!\langle \mathcal{P} \rangle\!\rangle^M$ with $M \geq M_\varphi$, we get that the set of verification objectives in the real-time verification procedure is the same as the set of verification objectives in the symbolic verification procedure. Moreover real-time rules can be mimicked by symbolic rules. Of course while checking for satisfiability, we do not have states of a system. So we will use the procedure as above with "imaginary" states. For this to work we will need to keep the timing information in a form of regions.

Now we formalise the rules and the procedure above by defining the tableau system of rules for Event-Recording Logic following the one introduced for the μ-calculus [Koz82, Wal95] and other temporal logics [GM96, Gor99, LS01]. We claim that our tableau system of rules is different and simpler than the one proposed by Sorea [Sor02].

A tableau system of rules is a collection of rules. A *rule* is of the form

$$\frac{T_1\, T_2\, \ldots T_n}{T}$$

where T and T_i for every $i = 1..n$ are tuples made of a set of formulas (satisfiability objectives) and a region (timing context). The tuples over the lines of a rules (T_i for every $i = 1..n$) are called the *premises* of the rules and the tuple T below the line of a rule is called *conclusion* of the rule. A rule as above is interpreted as follows: verifying whether the satisfiability objectives in the conclusion are satisfiable from the timing context is reduced to checking if all (some) satisfiability objectives in the hypothesis are satisfiable in their corresponding timing context.

Definition 94 (The System of Tableau Rules) The system of tableau rules $\mathcal{S}^{\varphi}$ parametrised by a formula φ (or rather its binding function $\mathcal{D}_{\varphi}$) and the set of regions $\mathcal{R}eg_M$ is defined by:

$$\frac{\{ff\}; ff}{\{\varphi, \Gamma\}; ff}(ff_r)$$

$$\frac{\{\varphi_1, \Gamma\}; r \qquad \{\varphi_2, \Gamma\}; r}{\{\varphi_1 \vee \varphi_2, \Gamma\}; r}(\vee) \qquad \frac{\{\varphi_1, \varphi_2, \Gamma\}; r}{\{\varphi_1 \wedge \varphi_2, \Gamma\}; r}(\wedge)$$

$$\frac{\{\varphi(X), \Gamma\}; r}{\{\mu X.\varphi(X), \Gamma\}; r}(\mu) \qquad \frac{\{\varphi(X), \Gamma\}; r}{\{\nu X.\varphi(X), \Gamma\}; r}(\nu)$$

$$\frac{\{\varphi(X), \Gamma\}; r}{\{X, \Gamma\}; r}(reg) \quad \mathcal{D}_{\varphi}(X) = \sigma X.\varphi(X)$$

$$\frac{\varphi \cup \{\bigvee \Theta_j \,|\, (g,a) \to \Theta_j \in \Gamma,\, \Theta_j \neq \Theta_i\}; (g \wedge r\!\uparrow)[h_a := 0] \quad \text{for every} \begin{cases} (g,a) \to \Theta_i \in \Gamma, \\ \varphi \in \Theta_i \end{cases}}{\Gamma; r}(mod)$$

We remark that if g is rectangular then $(g \wedge r\!\uparrow)$ is a region (or is inconsistent). Thus, if we start with a rectangular formula then all time-contexts obtained by applications of the rules will be regions.

We give the intuitive idea behind the rule (mod). The conclusion $\Gamma; r$ of the rule (mod) is such that every formula in Γ is of the form $(g,a) \to\!\Theta$ where Θ is a set of formulas. Recall that $(g,a) \to\!\Theta \equiv \bigwedge_{\varphi \in \Theta} \langle g,a \rangle \varphi \wedge [g,a] \bigvee_{\varphi \in \Theta} \varphi$. Then the presence of existential modality requires that when the time elapses, the constraint occurring in a formula $(g,a) \to\!\Theta_i$ of Γ should be satisfied (when $\Theta_i \neq \emptyset$) and then every $\varphi \in \Theta_i$ should also be satisfied. We need something more though as Γ may contain a collection of formulas of the form $(g,a) \to\!\Theta_j$, for the same guarded-event (g,a). In this case we need when to check if $\varphi \in \Theta_i$ together with the disjunctions $\bigvee \Theta_j$ are satisfied.

Definition 95 (Tableau) A tableau for a rectangular formula φ from a region r^0 is a pair $\tau^{\varphi}_{r^0} = \langle \mathcal{T}, \mathcal{L} \rangle$, where $\mathcal{T} = \langle N, E \rangle$ is a tree, and $\mathcal{L}$ is a labeling function such that:

1. The root n^0 of $\tau^{\varphi}_{r^0}$ is labeled by $\{\varphi\}; r^0$
2. The sons of any node n are created and labeled according to the rules of system $\mathcal{S}^{\varphi}$. It is required the rule (*mod*) is applied only when no other rule is applicable.

Example: Let us present a fragment of the tableau for a formula

$$\varphi = (0 < h_a < 1, a) \rightarrow \{\varphi_1, \varphi_2\} \wedge (0 < h_a < 1, a) \rightarrow \{\varphi_3\} \wedge (0 < h_a < 1, b) \rightarrow \{\varphi_4\}$$

where, r_0 is the region satisfying the constraint $h_a = 0 \wedge h_b = 0$, r_1 is the region satisfying the constraint $h_a = 0 \wedge 0 < h_b < 1$ and, r_2 is the region satisfying the constraint $0 < h_a < 1 \wedge h_b = 0$

$$\frac{\{\varphi_1, \varphi_3\}; r_1 \qquad \{\varphi_2, \varphi_3\}; r_1 \qquad \{\varphi_3, \varphi_1 \vee \varphi_2\}; r_1 \qquad \{\varphi_4\}; r_2}{\frac{\{(0 < h_a < 1, a) \rightarrow \{\varphi_1, \varphi_2\}, (0 < h_a < 1, a) \rightarrow \{\varphi_3\}, (0 < h_a < 1, b) \rightarrow \{\varphi_4\}\}; r_0}{\frac{\{(0 < h_a < 1, a) \rightarrow \{\varphi_1, \varphi_2\}, (0 < h_a < 1, a) \rightarrow \{\varphi_3\} \wedge (0 < h_a < 1, b) \rightarrow \{\varphi_4\}\}; r_0}{\{(0 < h_a < 1, a) \rightarrow \{\varphi_1, \varphi_2\} \wedge (0 < h_a < 1, a) \rightarrow \{\varphi_3\} \wedge (0 < h_a < 1, b) \rightarrow \{\varphi_4\}\}; r_0}}}$$

□

If n is a node of the tableau and $\mathcal{L}(n) = \Gamma; r$, then $\mathcal{L}_{ERL}(n) = \Gamma$ and $\mathcal{L}_{\rho}(n) = r$ denote the *formula part* and the *timing part* of $\mathcal{L}(n)$.

We remark that the application of a rule at some node n depends both on some formula $\varphi \in \mathcal{L}_{ERL}(n)$ and the time context $\mathcal{L}_{\rho}(n)$. We say that the rule is *directed* by the tuple $\varphi; \mathcal{L}_{\rho}(n)$.

We remark that the tableau of a formula φ is a finite branching tree nodes of which are labeled on the finite alphabet $2^{sub(\varphi)} \times \mathcal{R}eg$.

Definition 96 A *modal node* is a node in which the rule (*mod*) is applied; A *disjunctive (resp.conjunctive) node* is a node in which the rule $\vee$ (resp $\wedge$) is applied. Since a modal node may has several successors, a (g, a)-*son* of a modal node is the son obtained by considering the guarded event (g, a) for some $\varphi \in \Theta_i$ with $(g, a) \rightarrow \Theta_i \in \Gamma$.

Example: In the example just above, the node labelled with

$$\{(0 < h_a < 1, a) \rightarrow \{\varphi_1, \varphi_2\} \wedge (0 < h_a < 1, a) \rightarrow \{\varphi_3\} \wedge (0 < h_a < 1, b) \rightarrow \{\varphi_4\}\}; r_0$$

is a conjunctive node. The node labelled with

$$\{(0 < h_a < 1, a) \rightarrow \{\varphi_1, \varphi_2\}, (0 < h_a < 1, a) \rightarrow \{\varphi_3\}, (0 < h_a < 1, b) \rightarrow \{\varphi_4\}\}; r_0$$

is a modal node which has three $(0 < h_a < 1, a)$-sons and one $(0 < h_a < 1, b)$-son. □

Definition 97 (Choice node, near to) A *choice node* is a root node or a son of a modal node. A node m is *near* to a node n if and only if there is a path from n to m in a tableau without an application of the rule (*mod*) in-between.

We remark that the root of a tableau can be a choice node and a modal node. A leaf node, a modal node, or a disjunctive node, or a conjunctive node can also be a choice node.

3.3.2 Semantics of Tableau

A path in a tableau represents of partial task in the satisfiability checking of a set of formulas. Given $\Gamma_1; r_1$ and $\Gamma_2; r_2$ two consecutive nodes of the tableau, such that $\Gamma_2; r_2$ derived from $\Gamma_1; r_1$, there is a formula $\varphi_1 \in \Gamma_1$ that has been reduced according to an appropriate rule and there is a formula $\varphi_2 \in \Gamma_2$ which is one of the results of the reduction of φ_1. Such a relation between formulas in the premises and formulas in the conclusion keeps track of decomposition of each formula. We follow [JW95, NW96] and we formalise this relation by defining traces.

Definition 98 (Trace) Given a path π of $\tau^{\varphi}_{r_0} = \langle \mathcal{T}, \mathcal{L} \rangle$, a *trace* on π will be a function $\mathtt{F}$ which assigns a tuple made of a formula and a region to each node in some initial segment of π, according to the rules applied for the construction of π. $\mathtt{F}_{ERL}$ and $\mathtt{F}_{\rho}$ denote the *formula part* and the *timing part* of $\mathtt{F}(n)$. We require that $\mathtt{F}$ satisfies the following condition: let n be the successor of m on π then

1. if the rule applied m is not (mod) and it is not directed by $\mathtt{F}(m)$ then $\mathtt{F}(m) = \mathtt{F}(n)$;
2. if the rule applied at the node m is not (mod) but it is directed by $\mathtt{F}(m)$ then $\mathtt{F}_{\rho}(n) = \mathtt{F}_{\rho}(m)$ and $\mathtt{F}_{ERL}(n)$ is one of the results of the application the rule in m.
3. if the rule (mod) is applied at m and the son $n \in \pi$ of m is labeled by $\varphi \cup \{\bigvee \Theta_j : (g, a) \rightarrow \Theta_j \in \Gamma, \Theta_j \neq \Theta_i\}; r\uparrow \wedge g[h_a := 0]$ for some $(g, a) \rightarrow \Theta_i \in L_{ERL}(m)$ and $\varphi \in \Theta_i$ then $\mathtt{F}(n) = \varphi; r\uparrow \wedge g[h_a := 0]$ if $\mathtt{F}(m) = (g, a) \rightarrow \Theta_i; r$ and $\mathtt{F}(n) = \bigvee \Theta_j; r\uparrow \wedge g[h_a := 0]$ if $\mathtt{F}(m) = (g, a) \rightarrow \Theta_j; r$.

In order to establish whether a formula is satisfiable are not, we distinguish "good trace" from "bad trace". Intuitively, a "bad trace" is something that cannot appear in a correct execution of the verification procedure sketched on page 73. We also consider "good path" and "bad path". Again a "bad path" is a path that cannot occur during correct execution of the verification procedure. To characterise "good" and "bad" paths we need to consider finite and infinite paths.

A finite trace is "good" if it ends in a tuple the formula part of which is tt or $(g, a) \rightarrow \emptyset$. Otherwise it is "bad".

If a trace is infinite, there is are fixpoint variables that are infinitely often regenerated. Following [Koz82, JW95], we give the definition of variable regeneration.

Definition 99 A variable X is regenerated on a trace $\mathtt{F}$ of some path if and only if for some m and its son n on the path $\mathtt{F}_{ERL}(m) = X$ and $\mathtt{F}_{ERL}(n) = \psi(X)$ with $\mathcal{D}_{\varphi}(X) = \sigma X.\psi(X)$.

As stated later in Theorem 87, the "goodness" of a trace may depend on the nature of variables that are infinitely often regenerated. As introduced in the above examples, "good" and "bad" traces depends on the order between variables and the nature of the oldest variable that is infinitely often regenerated.

We formalise the notion of "good" and "bad" traces by defining μ-traces.

Definition 100 (μ-trace) A *μ-trace* is an infinite trace on which the oldest variable regenerated infinitely often is a *μ-variable*; or a finite trace, ending with a tuple the formula part of which contains *ff*.

So a "bad" trace is a μ-trace and a "good" is a trace that is not "bad". It follows that a "bad" path is a path that contains a μ-trace. We call such a path a μ-path. A "good" path is a path that does not contains a μ-trace.

Now that we have formally defined "good" paths and "bad" paths, we look for a distribution of "good" paths in the tableau in order to decide whether a set of formulas is satisfiable or not.

Definition 101 (Pre-model) A *pre-model* $\mathcal{PM}$ is a fragment of a tableau $\tau^{\varphi}_{r^0}$ satisfying the following conditions:

- The root of $\tau^{\varphi}_{r^0}$ belongs to $\mathcal{PM}$.
- If a disjunctive node belongs to $\mathcal{PM}$, then only one of its sons belongs to $\mathcal{PM}$.
- If a modal node belongs to $\mathcal{PM}$, then all its sons belong to $\mathcal{PM}$.
- There is no path with a μ-trace in $\mathcal{PM}$.

Definition 102 (refutation) A *refutation* $\mathcal{RF}$ is a fragment of a tableau $\tau^{\varphi}_{r^0}$ satisfying the following conditions:

- The root of $\tau^{\varphi}_{r^0}$ belongs to $\mathcal{RF}$.
- If a disjunctive node belongs to $\mathcal{RF}$, then all its sons belongs to $\mathcal{RF}$.
- If a modal node belongs to $\mathcal{RF}$, then at most one of its sons belong to $\mathcal{RF}$.
- There is a μ-trace on every path of $\mathcal{RF}$.

In the next section we show the following theorem.

Theorem 103 *A guarded rectangular formula φ is satisfiable if and only if there exists a pre-model for φ.*

3.3.3 Satisfiability Results

In this section, we present a proof of Theorem 103. As remarked before we can restrain to rectangular formulas.

As formulas may contain many fixpoint operators, then we need a structure to handle the variation of ordinals associated to each fixpoint operator after a computation step. For that purpose, we consider the notion of signature (also see [Wal95]).

Definition 104 (Signature, μ-signature, ν-signature) A *signature* $\mathtt{sig} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ is a sequence of ordinals value of which depends on a state. We distinguish μ-signature from ν-signature that we simply call signature when it is clear from the context.

Let a formula ψ without free variables, if $\mathcal{S}, s \vDash \langle\!\langle \psi \rangle\!\rangle_{\mathcal{D}_\varphi}$ then, ψ has the *μ-signature* ${}^{\mu}\mathtt{sig}(\psi, s) = (\alpha_1, \ldots \alpha_{d^\mu})$ in s if ${}^{\mu}\mathtt{sig}(s, \psi)$ is the least (in lexicographical order) sequence of

ordinals such that $\mathcal{S}, s \vDash \langle\!|\psi|\!\rangle_{\mathcal{D}'_\varphi}$ where $\mathcal{D}'_\varphi$ is obtained from the binding function $\mathcal{D}_\varphi$ by changing definitions of X_i (for $i = 1, \ldots, d^\mu$) from $\mathcal{D}_\varphi(X_i) = \nu X_i.\varphi_i(X_i)$ to $\mathcal{D}'_\varphi(X_i) = \nu^{\alpha_i} X_i.\varphi_i(X_i)$.

If $\mathcal{S}, s \nvDash \langle\!|\psi|\!\rangle_{\mathcal{D}_\varphi}$ then, ψ has the *ν-signature* ${}^\nu\mathtt{sig}(\psi, s) = (\alpha_1, \ldots \alpha_{d^\nu})$ in s if ${}^\nu\mathtt{sig}(s, \psi)$ is the least (in lexicographical order) sequence of ordinals such that $\mathcal{S}, s \nvDash \langle\!|\psi|\!\rangle_{\mathcal{D}'_\varphi}$ where $\mathcal{D}'_\varphi$ is obtained from the binding function $\mathcal{D}_\varphi$ by changing definitions of Y_i (for $i = 1, \ldots, d^\nu$) from $\mathcal{D}_\varphi(Y_i) = \mu Y_i.\varphi_i(Y_i)$ to $\mathcal{D}'_\varphi(Y_i) = \mu^{\alpha_i} Y_i.\varphi_i(Y_i)$.

In the definition of the notions of ${}^\mu$-signature and ${}^\nu$-signature just above $\mathcal{S}$ is either a $(\Sigma \cup \mathit{Val}_\Sigma)$-labelled transition system representing the semantics of a timed process, and in this case $\vDash$ is the relation $\vDash_t$; or a $(\Sigma \cup \mathit{Gds})$-labelled transition system representing the semantics of a timed process, and in this case $\vDash$ is the relation $\vDash_g$.

Lemma 105 (μ-Signature) Let ${}^\mu\mathtt{sig}(\varphi, s)$ the signature of φ at s, it is true that:

- ${}^\mu\mathtt{sig}(\varphi_1 \wedge \varphi_2, s) = \max\{{}^\mu\mathtt{sig}(\varphi_1, s), {}^\mu\mathtt{sig}(\varphi_2, s)\}$
- ${}^\mu\mathtt{sig}(\varphi_1 \vee \varphi_2, s) = {}^\mu\mathtt{sig}(\varphi_1, s)$ or ${}^\mu\mathtt{sig}(\varphi_1 \vee \varphi_2, s) = {}^\mu\mathtt{sig}(\varphi_2, s)$
- for all $\varphi \in \Theta$, there is s' such that $s \xrightarrow{\hat{g}, a} s'$ and ${}^\mu\mathtt{sig}(\varphi, s') \leq {}^\mu\mathtt{sig}((\hat{g}, a) \rightarrow \Theta, s)$; and for every s' such that there is a transition from $s \xrightarrow{\hat{g}, a} s'$, we have ${}^\mu\mathtt{sig}(\bigvee \Theta, s') \leq^\mu \mathtt{sig}((\hat{g}, a) \rightarrow \Theta, s)$
- if X_i is the $i - th$ variable of $\mathcal{D}_\varphi$ and $\mathcal{D}_\varphi(X_i) = \mu X_i \varphi(X_i)$, then the prefixes of length $i - 1$ of ${}^\mu\mathtt{sig}(\mu X_i.\varphi(X_i), s)$ and ${}^\mu\mathtt{sig}(\varphi(X), s)$ are equal
- ${}^\mu\mathtt{sig}(\nu X.\varphi(X), s) = {}^\mu\mathtt{sig}(\varphi(X), s)$ where $\mathcal{D}_\varphi(X) = \nu X.\varphi(X)$
- if $\mathcal{D}_\varphi(Y) = \mu Y.\varphi(Y)$, then ${}^\mu\mathtt{sig}(Y, s) > {}^\mu\mathtt{sig}(\varphi(Y), s)$
- if $\mathcal{D}_\varphi(Y) = \nu Y.\varphi(Y)$, then ${}^\mu\mathtt{sig}(Y, s) = {}^\mu\mathtt{sig}(\varphi(Y), s)$

Proof
The case of $(\hat{g}, a) \rightarrow \Theta$ is a generalisation of the cases of $\varphi_1 \wedge \varphi_2$ and $\varphi_1 \vee \varphi_2$ which themselves are immediate.

Considering the last cases, we suppose that $\mathcal{S}, s \vDash \langle\!|X_i|\!\rangle_{\mathcal{D}_\varphi}$ with $\mathcal{D}_\varphi(X_i) = \mu X_i.\psi_i(X_i)$. X_j occurs in $\psi_i(X_i)$ implies that $X_i \leq_\varphi X_j$ and X_j is free $\psi_i(X_i)$. Let ${}^\mu\mathtt{sig}(X_i, s) = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ and $\mathcal{D}'$ obtained from $\mathcal{D}_\varphi$ by changing definitions of X_i (for $i = 1, \ldots, d^\mu$) from $\mathcal{D}_\varphi(X_i) = \mu X_i.\psi_i(X_i)$ to $\mathcal{D}'_\varphi(X_i) = \mu^{\alpha_i} X_i.\psi_i(X_i)$.

It follows from the definition of the signature that $\mathcal{S}, s \vDash_g \mu^{\alpha_i} X_i.\psi(X_i)$. This implies that α_i is a successor ordinal. It follows that $\mathcal{S}, s \vDash_g \psi(\mu^{\alpha_i - 1} X.\psi(X_i))$. This means that the signature of $\psi(\mu^{\alpha_i - 1} X_i.\psi(X_i))$ at s is $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_i - 1, \alpha'_{i+1}, \ldots, \alpha'_{d^\mu})$ and is smaller than $\mathtt{sig}(W_i, s)$. The difference occurs at the position i. □

Lemma 106 (ν-Signature) Let ${}^\nu\mathtt{sig}(\varphi, s)$ the signature of φ at s, the following assertions hold:

- ${}^\nu\mathtt{sig}(\varphi_1 \vee \varphi_2, s) = \max\{{}^\nu\mathtt{sig}(\varphi_1, s), {}^\nu\mathtt{sig}(\varphi_2, s)\}$

- ${}^{\nu}\mathtt{sig}(\varphi_1 \wedge \varphi_2, s) = {}^{\nu}\mathtt{sig}(\varphi_1, s)$ or ${}^{\nu}\mathtt{sig}(\varphi_1 \wedge \varphi_2, s) = {}^{\nu}\mathtt{sig}(\varphi_2, s)$
- Either exists $\varphi \in \Theta$ such that for all s', if $s \xrightarrow{\hat{g},a} s'$ for some $\hat{g}$, then we have ${}^{\nu}\mathtt{sig}(\varphi, s') \leq^{\nu} \mathtt{sig}((\hat{g}, a) \to \Theta, s)$; or there is s' such that $s \xrightarrow{\hat{g},a} s'$, and ${}^{\nu}\mathtt{sig}(\bigvee \Theta, s') \leq^{\nu} \mathtt{sig}((g, a) \to \Theta, s)$.
- If X_i is the $i-th$ variable of $\mathcal{D}_\varphi$ and $\mathcal{D}_\varphi(X_i) = \mu X_i \varphi(X_i)$, then the prefix of length $i-1$ of ${}^{\nu}\mathtt{sig}(\mu X_i.\varphi(X_i), s)$ and ${}^{\nu}\mathtt{sig}(\varphi(X), s)$ are equal.
- ${}^{\nu}\mathtt{sig}(\nu X.\varphi(X), s) = {}^{\nu}\mathtt{sig}(\varphi(X), s)$ where $\mathcal{D}_\varphi(X) = \nu X.\varphi(X)$
- If $\mathcal{D}_\varphi(Y) = \mu Y.\varphi(Y)$, then ${}^{\nu}\mathtt{sig}(Y, s) = {}^{\nu}\mathtt{sig}(\varphi(Y), s)$
- If $\mathcal{D}_\varphi(Y) = \nu Y.\varphi(Y)$, then ${}^{\nu}\mathtt{sig}(Y, s) > {}^{\nu}\mathtt{sig}(\varphi(Y), s)$

Proof
Dual to Lemma 105. □

Proposition 107 Any tableau for a formula φ contains either a pre-model or a refutation.

Proof
(Sketch) The proof of this result is the same as the proof of a similar result in the setting of the for μ-calculus [Wal95]. The proof uses some results on two player parity games. One defines a two player parity game and shows that a player has a winning strategy if and only if there is a pre-model for φ and its adversary has a winning strategy if and only if there is a refutation for φ. The conclusion comes from the fact that two player parity games are determined (see Theorem 11). □

Proposition 108 Any tableau of a satisfiable rectangular guarded formula φ contains a pre-model for φ.

Proof
The proof follows the ones in [Koz82, Wal95].
Suppose that there is timed process $\mathcal{P}$ such that $\langle\![\mathcal{P}]\!\rangle^M, s^0 \vDash_g \varphi$.

Let $\tau^{\varphi}_{r^0}$ a tableau for φ. We aim at constructing a pre-model $\mathcal{PM}$ for φ which is in fact a sub tree of $\tau^{\varphi}_{r^0}$. It means to choose the nodes of $\tau^{\varphi}_{r^0}$ that we will include in $\mathcal{PM}$. Of course, the root of $\tau^{\varphi}_{r^0}$ will be included in $\mathcal{PM}$. We assign to each node n that has been included in $\mathcal{PM}$, a state $s_n \in S$ such that $\langle\![\mathcal{P}]\!\rangle^M_{reg}, s_n \vDash_g \langle\!|\psi|\!\rangle_{\mathcal{D}_\varphi}$ for every $\psi \in \mathcal{L}_{ERL}(n)$. This assignment will be done through the so-called *marking relation* $M : N \to S$. So we will have

(1) if $s_n = M(n)$ then $\langle\![\mathcal{P}]\!\rangle^M, s_n \vDash_g \langle\!|\psi|\!\rangle_{\mathcal{D}_\varphi}$ for every $\psi \in \mathcal{L}_{ERL}(n)$.

We set $s^0 = M(n^0)$ where n^0 is the root of $\tau^{\varphi}_{r^0}$.This satisfies (1).

Now, assume that a node n has been included in $\mathcal{PM}$ with $s_n = M(n)$. We use the rules for the tableau to select the next nodes that we include in $\mathcal{PM}$. The selection works as follows:

- The only son of some node n, marked with s_n, on which an unary rule ($\not\!\!f\!f_r, \wedge, reg, \mu$, or ν) was applied is included in $\mathcal{PM}$; this son is marked with s_n.

- If n is a disjunctive node, then s_n is put into the marking of the son for which it has the least signature. Such a son exists by Lemma 105.

- If n is a modal node, then we add all the sons of n in $\mathcal{PM}$. Each son n' of n is the result of the reduction of a formula $(\hat{g}, a) \to \Theta \in \mathcal{L}_{ERL}(n)$ with respect to some $\psi \in \Theta$. Set $M(n') = s_{n'}$ where $s_{n'}$ is such that $s_n \xrightarrow{\hat{g},a} s_{n'}$, and ${}^{\mu}\mathtt{sig}((\hat{g}, a) \to \Theta, s_n) \geq^{\mu} \mathtt{sig}(\psi, s_{n'})$. Such a configuration exists by Lemma 105.

By Property (1) above, it is obvious that no leaf of $\mathcal{PM}$ contains *ff*. It remains to show that the tree we have constructed does not have an infinite path with a μ-trace.

Now, assume that there is an infinite path π that has a μ-trace on it. Then, there is the oldest μ-variable X_i infinitely often regenerated along the trace. According to Lemma 105, from the point when no variable older that X_i is regenerated, μ-signatures of formulas on that trace never increase on positions $1, \ldots, i-1$. Then maximal signature of formulas on the trace considered up to position i never increases and decreases every time X_i is regenerated. This is a contradiction because sequences of ordinals of bounded length are well-ordered. □

From the definition of the system of tableau rules, applying a rule different from (*mod*) and $\vee$ to a node of a tableau generates a unique successor. In a pre-model we choose only one son of a disjunctive node and all the sons of a modal node. It follows that in a pre-model, a node with more that one successor is a modal node. Given a node n of $\mathcal{PM}$ we denote $des(n)$ the closest descendant of n or n itself in that is either a modal node or a leaf.

Definition 109 (sharply guarded model for a pre-model) Given a pre-model $\mathcal{PM} = \langle K, L \rangle$, the *sharply guarded model* based on $\mathcal{PM}$ is the timed process $\mathcal{S} = \langle S, \Sigma, s^0, \Delta_{\mathcal{S}} \rangle$ such that:

1. S consists of all nodes of $\mathcal{PM}$ that are either leaves, or modal nodes.

2. $(s, \hat{g}, a, s') \in \Delta_{\mathcal{S}}$ if there is in $\mathcal{PM}$ a son n of s with $des(n) = s'$, such that the label of n was obtained from the label of s by reducing a formula of the form $(\hat{g}, a) \to \Theta$.

We remark that the maximal constant that occurs in the sharply guarded model is smaller or equal to the maximal constant that occurs in the formula.

Proposition 110 Formula φ is satisfiable in the sharply guarded model associated to a pre-model of φ.

Proof
The proof is dual to the one of Proposition 108. We will assume that $\mathcal{PM}$ is a pre-model for φ and φ is not satisfiable in the sharply-guarded model $\mathcal{P}$ associated to $\mathcal{PM}$. Then, we obtain a contradiction.

If φ is not satisfiable in $\mathcal{P}$ then $\langle\!\langle \mathcal{P} \rangle\!\rangle^M_{reg}, s^0 \not\models_g \varphi$. Recall that states of $\langle\!\langle \mathcal{P} \rangle\!\rangle^M_{reg}$ are the leaves or the modal nodes of $\mathcal{PM}$. Using the assumption that $\langle\!\langle \mathcal{P} \rangle\!\rangle^M_{reg}, s^0 \not\models_g \varphi$, we show that $\mathcal{PM}$ contains a path π with a μ-trace $\mathtt{F} = \{\varphi_n; r_n\}_{n \in \pi}$. The expected path π and the μ-trace are constructed as follows:

- π starts at n^0 and $\varphi_{n^0} = \varphi$. By the hypothesis, $\langle\!\langle \mathcal{P} \rangle\!\rangle^M_{reg}, des(n^0) \not\Vdash_g \varphi$ since n^0 is the node of $\mathcal{PM}$ associated to the initial configuration of $\langle\!\langle \mathcal{P} \rangle\!\rangle^M_{reg}$.
- Now assuming that we have constructed $\mathtt{F}$ up to the node $\varphi_m; r \in \mathcal{L}_{ERL}(m) \times \mathcal{L}_\rho(m)$ such that $\langle\!\langle \mathcal{P} \rangle\!\rangle^M_{reg}, des(m) \not\Vdash_g \langle\!\langle \varphi_m \rangle\!\rangle_{\mathcal{D}_\varphi}$. The formula of the next tuple (the timing part is obvious) is selected as follows:
  1. If m is not a modal node, then the only son m' of m is such that
     - $\varphi_{m'} = \varphi_m$ if φ_m was not reduced by the rule.
     - $\varphi_{m'} = \varphi_1$ if $\varphi_m = \varphi_1 \wedge \varphi_2$ and ${}^\nu\mathtt{sig}(\varphi_m, des_m) \geq {}^\nu\mathtt{sig}(\varphi_1, des_m)$;
     - $\varphi_{m'} = \varphi_2$ if $\varphi_m = \varphi_1 \wedge \varphi_2$ and not above;
     - $\varphi_{m'}$ is the formula that occurs in $\mathcal{L}_{ERL}(m')$ if $\varphi_m = \varphi_1 \vee \varphi_2$. Observe that the choice is directed by $\mathcal{PM}$.
     - In the other sub cases: $\mathit{ff}_r, \nu, \mu, reg$, just take the resulting formula as the next element of the trace.
  2. If m is a modal node and $\varphi_m = (\hat{g}, a) \rightarrow\!\Theta$ then
     - either there is $\psi \in \Theta$, such that for every son m' of m and $s \xrightarrow{\hat{g},a} s'$ with $des(m') = s'$ we have $\langle\!\langle \mathcal{P} \rangle\!\rangle^M_{reg}, s' \not\Vdash_g \langle\!\langle \psi \rangle\!\rangle_{\mathcal{D}_\varphi}$ and ${}^\nu\mathtt{sig}(\psi, s') \leq {}^\nu\mathtt{sig}((\hat{g}, a) \rightarrow\!\Theta, s)$. In this case, just take $\varphi_{m'} = \psi$;
     - or, there is a son m' of m with $des(m) = s'$, and $s \xrightarrow{\hat{g},a} s'$ such that $\langle\!\langle \mathcal{P} \rangle\!\rangle^M_{reg}, s' \not\Vdash_g \bigvee \Theta$ and ${}^\nu\ \mathtt{sig}(\bigvee \Theta, s') \leq^\nu \mathtt{sig}((\hat{g}, a) \rightarrow \Theta, s)$. In this case consider m' and set $\varphi_{m'} = \bigvee \Theta$ or $\gamma \in \Theta$ depending on which one appears in $\mathcal{L}(m')$.

There are two cases:

1. The above trace is finite. If the last element is ff, there is a contradiction with that the sharply guarded model correspond to the pre-model; and a pre-model does not contain a μ-trace, in particular it does not contain a trace that ends with a tuple the formula part of which is ff. If the last tuple does not contain the formula ff, it contains a formula ψ of the form $(\hat{g}, a) \rightarrow \emptyset$ or tt; from the definition of the sharply guarded model, it follows that $\langle\!\langle \mathcal{P} \rangle\!\rangle^M_{reg}, des(m) \models_g \langle\!\langle \psi \rangle\!\rangle_{\mathcal{D}_\varphi}$, this is in contradiction with the third item of Lemma 106, because we assumed that $\langle\!\langle \mathcal{P} \rangle\!\rangle^M_{reg}, des(m)$ is not satisfied the formula assigned to m. In fact, if the last tuple occurs in the node m and contains the formula:
   - tt, then $\langle\!\langle \mathcal{P} \rangle\!\rangle^M_{reg}, des(m) \models_g tt$ and we get a contradiction with the hypothesis.
   - $(\hat{g}, a) \rightarrow\!\emptyset$, then m is either a modal node or a leaf. In both cases, m does not have a son n obtained from it by reducing a formula $(\hat{g}, a) \rightarrow\!\Theta$ with respect to some Θ and $\varphi_c \in \Theta$ as otherwise the that could not end with $(\hat{g}, a) \rightarrow\!\emptyset$. As $(\hat{g}, a) \rightarrow\!\emptyset \equiv [\hat{g}, a]\mathit{ff}$, by definition of $\models_g$ we get the contradiction with the hypothesis that $\langle\!\langle \mathcal{P} \rangle\!\rangle^M_{reg}, des(m) \not\Vdash_g (\hat{g}, a) \rightarrow\!\emptyset$.
2. If the trace is infinite, the only way to have an infinite trace is to have a regeneration of a fixpoint variable. It cannot be a μ-variable as we are in a pre-model. Hence it must be a ν-variable. As by Lemma 106 the ν-signature decreases along the constructed trace, this is impossible.

□

3.3.4 Complexity Issues

We have reduced the satisfiability of a formula to the existence of a pre-model in its tableau. Then the complexity for the satisfiability of a formula is the same as the complexity of checking the existence of a pre-model in its tableau. The procedure for checking the existence of a pre-model in the tableau is the same as the procedure for checking the existence of a pre-model in the tableau for μ-calculus formula which is exponential in the size of the formula. In the case of ERL, our algorithm is also exponential in the size of the formula.

3.4 Comparison With Earlier Works

In the section we compare results in previous sections with Sorea's results on ERL. In [Sor02] Sorea proposed decision procedures for the model-checking and satisfiability of ERL formulas. These procedures were supposed to work even for general constraints (diagonal constraints were allowed). The Sorea [Sor02] procedure for the satisfiability problem of ERL is based on a tableau system of rules. Sequents in rules are couples, made of a set of formulas and a timing context. Timing context are represented by zones. In that tableau system, one particular rule could be interpreted in two ways. One way is wrong as it forbids the "division" of the time satisfying a constraint in the existential modality. The second interpretation is correct as it enables the "division" of the time. Nevertheless, the procedure for the satisfiability is not correct. Recall that the satisfiability procedure requires to check the existence of a "good" fragment in the tableaux. Indeed, the application of a rule in some node of the tableau depends on the consistence of the zone in that node and the rule uses the time elapse operation on zones. Normally, labelling of nodes will range over an infinite set and the satisfiability procedure may not terminate. To ensure finite set of labels for the nodes of a tableau, Sorea has proposed to approximate zones. As diagonal constraints were allowed, the procedure of Sorea declares some fragments of tableaux "good" while in reality they are not.

3.4.1 Sorea's Semantics for Timed Process and ERL Formulas

Let us briefly comment the Sorea' semantics for timed process and the relation with the semantics in Definition 66. Sorea's semantics for timed processes is standard. They Sorea's semantics of a timed process is a $(\mathbb{R}^+ \times \Sigma)$-LTS where there is transition $(p, v) \xrightarrow{t,a} (p', v + t[h_a := 0])$ whenever there is a transition $p \xrightarrow{g,a} p'$ and a delay $t \in \mathbb{R}^+$ such that $v+t \in [\![g]\!]$. That representation for Sorea's semantics for timed processes is isomorphic to the representation in which delays on transitions are replaced with valuations and defined in such a way that there is transition $(p, v) \xrightarrow{v+t,a} (p', v + t[h_a := 0])$ whenever there is a transition $p \xrightarrow{g,a} p'$ and a delay $t \in \mathbb{R}^+$ such that $v + t \in [\![g]\!]$. This latter representation is a $(Val_\Sigma \times \Sigma)$-LTS.
Semantics in Definition 66 is a $(Val_\Sigma \cup \Sigma)$-LTS. Using the notation just after Definition 66, one can observe that, there is a transition $(p, v) \xrightarrow{v+t,a} (p', v + t[h_a := 0])$ in the Sorea's semantics for a timed process $\mathcal{P}$ if and only if there is a transition $(p, v) \xrightarrow{v+t,a} (p', v + t[h_a := 0])$ in $[\![\mathcal{P}]\!]$. The same remark holds for representations that label transitions with constraints. Due to the relation between Sorea's semantics and our semantics for timed processes, the Sorea's interpretation of an ERL formula over a timed process is exactly the same as our interpretation of the formula over the timed process.

3.4.2 Sorea's Tableau System of Rules

Let us first recall the tableau system of rules of Sorea [Sor02]. Let $\varphi = \langle g, a\rangle\psi$ be an ERL formula, and Γ be a set of formulas, such that each formula in Γ is a variable X, or a formula of the form $\langle g', a'\rangle\psi'$ or $[g', a']\psi'$ for some constraint g', and event a'. Let the set $tob(g, a, \Gamma)$ be defined by:

$$tob(g, a, \Gamma) = \{[g', a]\psi \in \Gamma \mid [\![g \wedge g']\!] \neq \emptyset\}$$

The tableau system of rules proposed by Sorea [Sor02] is presented below. A rule is made of some number of premises and a conclusion. The timing context is represented with a zone.

$$\frac{\{ff\}; ff}{\{\varphi, \Gamma\}; ff}(ff_Z)$$

$$\frac{\{\varphi_1, \Gamma\}; Z \qquad \{\varphi_2, \Gamma\}; Z}{\{\varphi_1 \vee \varphi_2, \Gamma\}; Z}(\vee) \qquad \frac{\{\varphi_1, \varphi_2, \Gamma\}; Z}{\{\varphi_1 \wedge \varphi_2, \Gamma\}; Z}(\wedge)$$

$$\frac{\{\varphi(X), \Gamma\}; Z}{\{\mu X.\varphi(X), \Gamma\}; Z}(\mu) \qquad \frac{\{\varphi(X), \Gamma\}; Z}{\{\nu X.\varphi(X), \Gamma\}; Z}(\nu)$$

$$\frac{\{\varphi(X), \Gamma\}; Z}{\{X, \Gamma\}; Z}(reg) \quad \mathcal{D}_\varphi(X) = \sigma X.\varphi(X)$$

$$\frac{\{\{\Gamma_{g'}; Z'_{g'} \mid g' \in G_g\} \mid \langle g, a\rangle\varphi \in \Gamma\}}{\Gamma; Z}(mod)$$

where

$$G_g = \bigcup_{J \subseteq tob(g,a,\Gamma)} \{g \wedge \bigwedge_{g' \in J} g' \wedge \bigwedge_{g' \notin J} \neg g'\}$$

denotes a set of all constraints included in g, and $\Gamma_{g'}; Z'_{g'}$ is defined by:

$$\Gamma_{g'}; Z'_{g'} = \begin{cases} \{\varphi\}; (Z{\uparrow} \wedge g')[h_a := 0]) & if \quad \Gamma = \emptyset \qquad or \quad tob(g, a, \Gamma) = \emptyset \\ \{\varphi\} \cup \Phi_{g'}; (Z{\uparrow} \wedge g'))[h_a := 0] & if \quad tob(g, a, \Gamma \neq \emptyset \end{cases}$$

with $\Phi_{g'} = \{\psi \mid [g'', a]\psi \in tob(g, a, \Gamma) \neq \emptyset$ and $g' \subseteq g''\}$.

3.4.3 Existential Modality May Cause Constraint Division

In the tableau system of rules above, it is not clear what happens in the rule (mod). For a given formula $\langle g, a\rangle\varphi$ in the conclusion of the rule (mod), we consider a set of constraints G_g which is such that the intersection of every constraint in it with g is consistent. As no precision is done in [Sor02] on the use of this rule when checking the satisfiability of a formula, there are two possible interpretations:

1. The first interpretation may consist to consider all the timed sequents in the set $\{\Gamma_{g'}; Z'_{g'} \mid g' \in G_g\}$.

2. The second may consist to consider only one sequent in that set.

The first interpretation gives incorrect result as it may be enough to consider only one timed sequent to ensure the satisfiability of the formula. For instance consider the example below.

Example: Let the formula $\varphi^0 = \langle 1 \leq h_a < 2, a\rangle tt \wedge [h_a = 1, a]ff \wedge \langle h_a = 1, b\rangle tt$. Consider the following sets of sub formulas of φ^0:

$$\begin{aligned}
\Gamma_0 &= \{\langle 1 \leq h_a < 2, a\rangle tt \wedge [h_a = 1, a]ff \wedge \langle h_a = 1, b\rangle tt\} \\
\Gamma_1 &= \{\langle 1 \leq h_a < 2, a\rangle tt, [h_a = 1, a]ff \wedge \langle h_a = 1, b\rangle tt\} \\
\Gamma_2 &= \{\langle 1 \leq h_a < 2, a\rangle tt, [h_a = 1, a]ff, \langle h_a = 1, b\rangle tt\} \\
\Gamma_3 &= \{tt, ff\} \\
\Gamma_4 &= \{tt\}
\end{aligned}$$

The formula φ^0 has two existential modal operators and one universal modal operator. According to the tableau system of rules, we should consider the sets $tob(1 \leq h_a < 2, a, \Gamma_2) = \{h_a = 1\}$, $tob(h_a = 1, a, \Gamma_2) = \{h_a = 1\}$, and the sets $G_{1 \leq h_a < 2} = \{h_a = 1, 1 < h_a < 2\}$. $G_{h_a = 1} = \{h_a = 1\}$,

Then the tableau for φ^0 starting from the Z^0 in which the value of $h_a = 1$ is depicted in Figure 12.

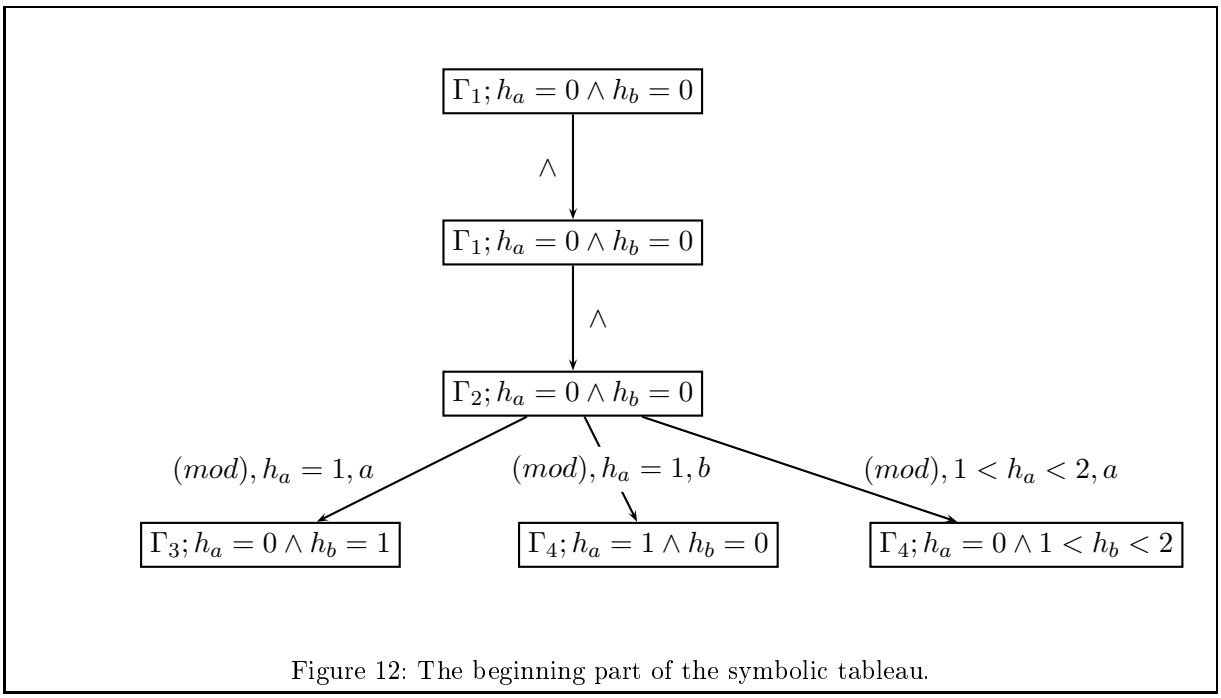

Figure 12: The beginning part of the symbolic tableau.

The tableau does not contain a disjunctive node. As Γ_3 contains ff and is included in the pre-model, the procedure of Sorea will assert that φ^0 does not have a model. This is not true since the timed process in Figure 13 is a model of φ^0.

□

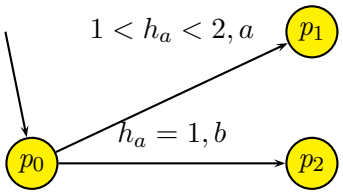

Figure 13: A timed process.

But, even the second interpretation gives incorrect result. As we show in the next subsection, the use of the approximation operation on zones to ensure finite set of labels of nodes of the tableau and then to ensure the termination of the satisfiability procedure, will make some paths "good" while they are not.

3.4.4 Zone Approach Is Not Correct

Consider the formula φ defined by $\varphi = \langle h_b \leq 3, c\rangle\varphi_1$ where,

$$\begin{aligned}
\varphi_1 &= \langle h_a \leq 3, a\rangle\varphi_2 \\
\varphi_2 &= \langle h_b = 3, b\rangle\varphi_3 \\
\varphi_3 &= \mu X.\varphi_4 \\
\varphi_4 &= \varphi_5 \vee \varphi_6 \\
\varphi_5 &= \langle h_a = 3, a\rangle\varphi_7 \\
\varphi_6 &= \langle h_a = 2, a\rangle\varphi_8 \\
\varphi_7 &= \langle h_a = 2, a\rangle\varphi_9 \\
\varphi_8 &= \langle h_b = 2, b\rangle X \\
\varphi_9 &= \langle h_b = 2, b\rangle\varphi_{11} \\
\varphi_{11} &= \langle h_a = 3, b\rangle\varphi_{12} \\
\varphi_{12} &= \langle h_b - h_a < 2 \wedge h_d - h_c > 2\rangle tt
\end{aligned}$$

Observe that φ has been inspired by the timed process in Figure 11 (Page 57) and each sub formula φ_i intends to describe the property of some state of that timed process.

A tableau for φ that uses the of rules of Sorea presented in Subsection 3.4.2 is presented in Figure 14. In this tableau every set of formulas in each timed sequent is a singleton. Every modal node has a single successor. There are disjunctive nodes $\{\varphi_4\}; Z_4^i$ where i is and integer. From a disjunctive node $\{\varphi_4\}; Z_4^i$ we can take the path $\{\varphi_4\}; Z_4^i \rightarrow \{\varphi_6\}; Z_6^i \rightarrow \{\varphi_8\}; Z_8^i \rightarrow \{X\}; Z_{10}^i \rightarrow \{\varphi_4\}; Z_4^{i+1}$ or take the path from $\{\varphi_4\}; Z_4^i$ to $\{tt\}; Z_{12}^i$. To check that φ is satisfiable, we must check that there is a pre-model in the tableau. As X is the single μ variable in φ, it must be regenerated only finitely times in a trace of the pre-model. As every node except disjunctive nodes has a single successor, the unique trace of the pre-model is of the form $\{\varphi\}; Z_0 \rightarrow \{\varphi_1\}; Z_1 \rightarrow \{\varphi_2\}; Z_2 \rightarrow \{\varphi_4\}; Z_4^0 \rightarrow \{\varphi_6\}; Z_6^0 \rightarrow \{\varphi_8\}; Z_8^0 \rightarrow \{X\}; Z_{10}^0 \rightarrow \{\varphi_4\}; Z_4^1 \rightarrow \cdots \rightarrow \{\varphi_4\}; Z_4^k \rightarrow \{\varphi_6\}; Z_6^k \rightarrow \{\varphi_8\}; Z_8^k \rightarrow \{X\}; Z_{10}^k \rightarrow \{\varphi_4\}; Z_4^{k+1} \rightarrow \{\varphi_5\}; Z_5^{k+1} \rightarrow \{\varphi_7\}; Z_7^{k+1} \rightarrow \{\varphi_9\}; Z_9^{k+1} \rightarrow \{\varphi_{11}\}; Z_{11}^{k+1}$ where k is the number of times the variable X is regenerated.

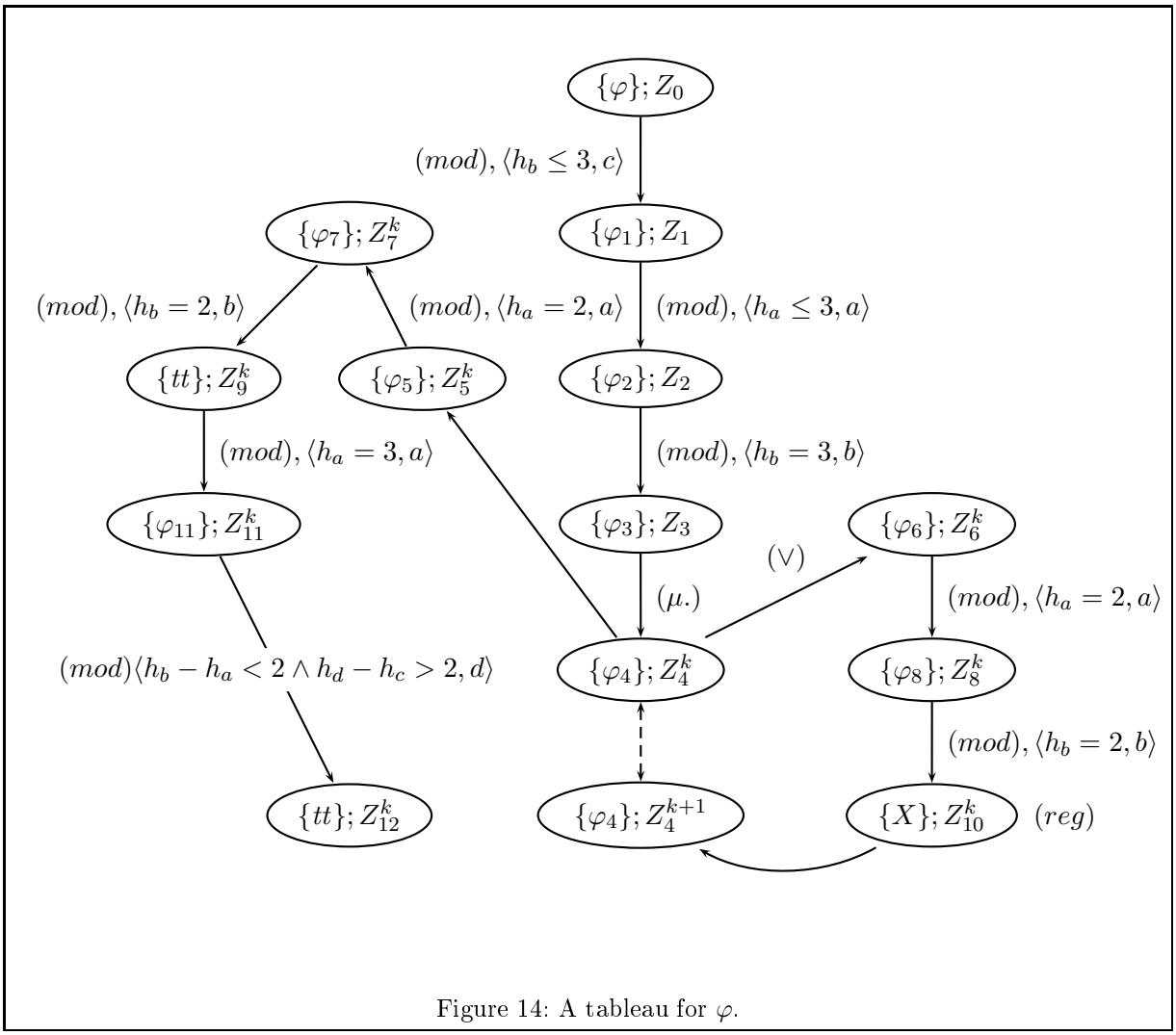

Figure 14: A tableau for φ.

It is not difficult to see that satisfiability of φ is reduced to the reachability problem of $\{tt\}; Z_{12}^{k+1}$. This problem has been discussed in Subsection 2.6.2 where we present an automaton that has the same structure as our tableau.

Following remarks we have done in Subsection 2.6.2, The zone Z_{11}^{k+1} is the following

$$\begin{cases} h_a = 0 \\ h_b \geq 1 \\ h_c \geq 2\gamma + 5 \\ h_d \geq 2\gamma + 6 \\ 2\gamma + 6 \leq h_a - h_d \leq 2\gamma + 8 \\ 1 \leq h_b - h_a \leq 3 \\ 2\gamma + 5 \leq h_c - h_a \leq 2\gamma + 8 \\ 2\gamma + 2 \leq h_c - h_b \leq 2\gamma + 5 \\ h_d - h_b = 2\gamma + 5 \\ 0 \leq h_d - h_c \leq 3 \end{cases}$$

and $Z_{11}^{k+1}\uparrow$ the zone obtained from Z_{11}^{k+1} by replacing the constraint $h_a = 0$ with $0 < h_a$. We remark that the bounds of diagonal constraints are not modified. It is clear that $Z_{11}^{k+1}\uparrow \wedge h_b - h_a < 2 \wedge h_d - h_c > 2$ is inconsistent as taking $h_b - h_a < 2$ implies (using constraints in $Z_{11}^{k+1}\uparrow$) that $h_d - h_c \leq 2$. Then the application of the rule (mod) in $\{\varphi_{11}\}; Z_{11}^{k+1}$ may produce the timed sequent $\{ff\}; ff$. That is enough to conclude that φ is not satisfiable.

In [Sor02], Sorea uses the normalisation (approximation) operator $Norm_M$ on zone that occurs in the tableau to ensure the termination of its tableau-based decision procedure for the satisfiability of formulas. When applying the normalisation operator at each node, Z_{11}^{k+1} will become the zone defined by the following constraints:

$$\begin{cases} h_a = 0 \\ h_b \geq 1 \\ h_c > k \\ h_d > k \\ h_a - h_d > k \\ 1 \leq h_b - h_a \leq 3 \\ h_c - h_a > k \\ h_c - h_b > k \\ h_d - h_b > k \\ 0 \leq h_d - h_c \leq 3 \end{cases}$$

For a sufficiently big k, $Z_{11}^{k+1}\uparrow \wedge h_b - h_a < 2 \wedge h_d - h_c > 2$ is consistent and $\{tt\}; Z_{12}^{k+1}$ is the last node of the path. As the unique trace is finite and ends with a tuple formula part of which is tt. We will get that φ is satisfiable, which is not correct.

3.5 Disjunctive Normal Form

The use of conjunctions in formulas and the alternation of fixpoint operators has required the use of set of formulas in timed sequent of the tableau leading to exponential algorithm for the satisfiability. Are there some kind of formulas for which sets of formulas do not need to be considered in timed sequents? If so, what is the expressive power of this kind of formulas? As we will see the answer to the first problem is yes for disjunctive normal form formulas that have the same expressive power as formulas in general form. In this section we consider the transformation of general ERL formulas into equivalent formulas in disjunctive normal form and we consider the satisfiability of later formulas. For that purpose we construct disjunctive formulas from tableau and we use the equivalence between tableaux of the disjunctive formula and the initial formula to show their equivalence.

3.5.1 Definition and Satisfiability Results

Let us first define disjunctive normal form for formulas.

Definition 111 (Disjunctive normal form) The set $\mathcal{F}_d$ of formulas in disjunctive normal form, is the smallest set defined by the following clauses:

1. Every variable is a disjunctive formula.

2. If $\varphi, \psi \in \mathcal{F}_d$ then $\varphi \vee \psi \in \mathcal{F}_d$; If moreover X does not occur in a sub formula of φ of the form $X \wedge \gamma$, then $\mu X.\varphi(X)$, $\nu X.\varphi(X) \in \mathcal{F}_d$.

3. Formula $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n \in \mathcal{F}_d$ provided that every φ_i is in $\{tt, ff\}$ or a formula of the form $\varphi_i = (\hat{g}_i, a_i) \rightarrow \Theta_i$ with $\Theta_i \subseteq \mathcal{F}_d$. We require that for any pair of guarded-events $(\hat{g}_i, a_i)$ and $(\hat{g}_j, a_j)$, $a_i = a_j$ implies that $\hat{g}_i \neq \hat{g}_j$.

We remark that modulo the order of application of the rule (*and*), disjunctive formulas have unique tableaux. Moreover on any infinite path there is one and only one infinite trace.

The proof of the following theorem uses similar argument to the proof of a similar result on the standard μ-calculus [JW95].

Theorem 112 *A closed disjunctive formula φ is satisfiable if and only if the formula ψ obtained from φ by replacing all occurrences of μ-variables by ff and all occurrences of ν-variables by tt is satisfiable.*

Proof
Let $\tau^{\varphi}_{r^0}$ and $\tau^{\psi}_{r^0}$ be the tableaux for φ and ψ. The tableau $\tau^{\psi}_{r^0}$ is a finite tree while the tableau $\tau^{\varphi}_{r^0}$ is not necessarily a finite tree. But observe that $\tau^{\varphi}_{r^0}$ can be seen as an unfolding of a graph obtained from $\tau^{\psi}_{r^0}$ by adding back edges from every node labelled with tt or ff to the unique node formula part of which contains the definition of the corresponding variable; that definition should be a μ-formula for all the nodes labelled with $\{ff\}$ with ff corresponding to the substitution of a μ-variable, and it should be a ν-formula for all the nodes labelled with $\{tt\}$ with tt corresponding to the substitution of a ν-variable. So, we can assume the existence of a surjective function $f : \tau^{\varphi}_{r^0} \rightarrow \tau^{\psi}_{r^0}$ that assigns to a node of $\tau^{\varphi}_{r^0}$ a unique corresponding node in $\tau^{\psi}_{r^0}$.
It is easy to show, using structural induction, that if φ is satisfiable, so is ψ.
Conversely, assuming that ψ is satisfiable, then ψ has a model. Let $\mathcal{P}$ be a model for ψ. From $\mathcal{P}$, we can build a pre-model $\mathcal{PM}_{\psi}$ for ψ and from $\mathcal{PM}_{\psi}$ we build a sub tree $\mathcal{PM}_{\varphi}$ of $\tau^{\varphi}_{r^0}$ containing any node n of $\tau^{\varphi}_{r^0}$ such that $f(n)$ belongs to $\mathcal{PM}_{\psi}$. It is easy to show that the resulting sub tree is a pre-model of $\tau^{\varphi}_{r^0}$ meaning that φ is satisfiable. □

In what follows, we will prove the equivalence between ERL formulas and ERL formulas in disjunctive normal form.

Theorem 113 (disjunctive normal form) *For every formula φ, there exists an equivalent disjunctive formula $\breve{\varphi}$ such that for any timed process $\mathcal{P}$, $\mathcal{P} \models \varphi$ if and only if $\mathcal{P} \models \breve{\varphi}$.*

3.5.2 Tableau Equivalence and Tableau With Back Edges

For the proof of Theorem 113, we will define the notion of equivalence between tableaux. We will show that the equivalence between tableaux of two formulas implies the equivalence of the meaning of formulas. We will introduce the notion of *tableau with back edges* which are a kind of graphs obtained from tableaux by cutting suffixes of some infinite paths and by adding conveniently back edges from the root of the suffixes that have been cut to one of their ancestors. Then given a formula, we will build from one of its tableau, a tableau with back edges. From that tableau with back edges, we will construct a formula in disjunctive form having a tableau equivalent to the tableau of the given formula.

Tableau equivalence

Definition 114 (tableau equivalence) Let τ_1 and τ_2 be two tableaux. The tableaux τ_1 and τ_2 are *equivalent* if and only if there is a bijective mapping $\mathcal{E}$ between choice nodes, modal nodes and leaf nodes of τ_1 and τ_2 such that :

1. $\mathcal{E}(n) = m$ implies that n is root of τ_1 and m is root of τ_2, or n and m are both disjunctive nodes or both modal nodes.
2. If n_1 is a descendant of n then $\mathcal{E}(n_1)$ is a descendant of $\mathcal{E}(n)$. Moreover if n_1 is a $(\hat{g}, a)$-son of n, then $\mathcal{E}(n_1)$ is a $(\hat{g}, a)$-son of n.
3. The set of literals in $\mathcal{L}_{ERL}(n)$ is equal to the set of literals of $\mathcal{L}_{ERL}(\mathcal{E}(n))$.
4. There is a μ-trace on a path π of τ_1 if and only if there is a μ-trace on the image of π under $\mathcal{E}$ in τ_2

Observation 114.1 If $\mathcal{E} : \tau^{\varphi}_{r^0} \to \tau^{\psi}_{r^0}$ is a function showing the equivalence of $\tau^{\varphi}_{r^0}$ and $\tau^{\psi}_{r^0}$ then $\mathcal{E}^{-1} : \tau^{\psi}_{r^0} \to \tau^{\varphi}_{r^0}$ is also a function showing the equivalence of $\tau^{\psi}_{r^0}$ and $\tau^{\varphi}_{r^0}$.

Proposition 115 If two guarded formulas have equivalent tableaux then, they admit the same set of models

Proof
Let φ and ψ two M-rectangular formulas. Let $\tau^{\varphi}_{r^0}$ and $\tau^{\psi}_{r^0}$ be the tableaux for φ and ψ that are equivalent. Then, there is a bijective mapping $\mathcal{E} : \tau^{\varphi}_{r^0} \to \tau^{\psi}_{r^0}$ showing the equivalence. We will show that for any M-rectangular timed process $\mathcal{P}$, state p and valuation Val, we have that $\mathcal{P}, p^0, Val \vDash \varphi$ if and only if $\mathcal{P}, p^0, Val \vDash \psi$.

If $\mathcal{P}, p^0, Val \vDash \varphi$ then, by Proposition 88 $[\![\mathcal{P}]\!]^M, (p^0, v^0), Val \vDash_g \varphi$. Under this assumption, we will exhibit a pre-model in $\tau^{\varphi}_{r^0}$. We use constructions similar to the ones in the proof of Proposition 108. We consider a marking $M : N \to P \times \mathcal{V}_\Sigma$ that satisfies: if $(p_n, v_n) \in M(n)$ then $\langle\!\langle \mathcal{P} \rangle\!\rangle^M_{reg}, (p_n, v_n) \vDash_g \mathcal{L}_{ERL}(n)$. We construct a pre-model $\mathcal{PM}$ of $\tau^{\varphi}_{r^0}$. Now consider the image $\mathcal{E}(\mathcal{PM})$ which maps a node n of $\mathcal{PM}$ to the node $\mathcal{E}(n)$ in $\tau^{\psi}_{r^0}$. Because $\tau^{\varphi}_{r^0}$ is equivalent to $\tau^{\psi}_{r^0}$, the literals in every node n of $\mathcal{PM}$ and in $\mathcal{L}_{ERL}(\mathcal{E}(n))$ are the same. There is a bijective function between the (g, a) sons of n and the (g, a)-sons of $\mathcal{E}(n)$ implying that a configuration (p, v) appears in $M(n')$ where n' is a son of n if and only it appears in $M(\mathcal{E}(n'))$. Finally $\mathcal{E}$ maps the root of $\tau^{\varphi}_{r^0}$ with the root of $\tau^{\psi}_{r^0}$ and $\mathcal{E}(\mathcal{PM})$ does not contain a μ-trace as $\mathcal{PM}$ does not contain a μ-trace. Because $\mathcal{E}(\mathcal{PM})$ is a pre-model of $\tau^{\psi}_{r^0}$, the formula ψ is satisfiable. Next, by using a contradiction argument, we show that $\mathcal{P}$ is also a model of ψ.
Assume that $\mathcal{P}$ is not a model of ψ, then $[\![\mathcal{P}]\!]^M, (p^0, v^0), Val \nvDash_g \varphi$. Then, using constructions similar to the ones in the proof of Proposition 110 we can show that $\mathcal{E}(\mathcal{PM})$ contains a path π containing a μ-trace and we immediately get a contradiction as we have shown that $\mathcal{E}(\mathcal{PM})$ is a pre-model and a pre-model does not contain a μ-trace.

If $\mathcal{P}, p^0, Val \vDash \psi$ then $\mathcal{P}, p^0, Val \vDash \varphi$ as $\mathcal{E}^{-1}$ is also tableaux equivalence. □

The converse of that proposition is not true; take for example the formulas $(\hat{g}, a) \rightarrow\{tt\} \wedge (\hat{g}, a) \rightarrow\{tt\}$ and $(\hat{g}, a) \rightarrow\{tt\}$.

Tableau with back edges Recall that nodes of a tableau have finitely many different labels. Then, an infinite path in a tableau has node labels that infinitely often occur in that path. For "good" paths, the ones that do not contain a μ-trace, we will cut a suffix starting at some node label of which occur infinitely often and we will add a back edge to some ancestor of that node equipped with the same label. We do the same for "bad" paths, the ones having a μ-trace. When adding a back edge, we will care that each path, among all the paths obtained by taking the back edge, corresponds to a path of the tableau with the same nature ("good" or "bad").

Let us take a tableau $\tau^{\varphi}_{r^0} = \langle \mathcal{T}, \mathcal{L} \rangle$ of a formula φ with respect to some region r^0.

Proposition 116 There is an automaton that distinguishes μ-traces from ν-traces in a tableau for φ.

Proof
Recall that a formula is of finite length and therefore uses a finite number of variables. The automaton is a Rabin automaton whose states are variables, and who is always in the state corresponding to the last variable read. Acceptance condition is a set of pairs of subsets of states such that a right member of a pair contains a μ-variable X; the left member contains all the ν-variables that are older than X. This automaton accepts a trace if and only if it is a μ-trace. □

Corollary 117 There is a deterministic parity ω-regular automaton which decides if a path contains a μ-trace.

Proof
In every node, a transition of the automaton is the disjunction of transitions of the automaton of Proposition 116 on each formula in that node. Such an automaton is a non-deterministic Rabin automaton which can be translated into an equivalent deterministic parity automaton. □

Form a tableau, we show how to build a tableau with back edges that preserves the nature of the path of the tableau.

Lemma 118 (back edge tableau) Given $\tau^{\varphi}_{r^0} = \langle \mathcal{T}, \mathcal{L} \rangle$ a tableau of a formula φ, it is possible to construct a finite tree with back edges $\overline{\tau} = \langle \overline{\mathcal{T}}, \overline{\mathcal{L}} \rangle$ satisfying the following conditions:

1. $\overline{\tau}$ unwinds τ,
2. Every node to which a back edge points can be assigned color red or green in such a way that for any infinite path from the unwinding of $\overline{\tau}$ we have : there is a μ-trace on the path if and only if the highest node of $\overline{\tau}$ through which the path goes infinitely often is colored *red*.

Proof
Consider a tableau τ for φ, clearly τ is finite branching and labelled over a finite alphabet. A path π of τ that contains a μ-trace is such that there is a highest node n in π from which a μ-variable is regenerated infinitely often in the same time-context. There is a deterministic ω-regular automaton with parity condition that separates paths that contain μ-trace from paths that do not. That automaton assigns a state to each node of τ. Formally, let $\mathcal{S} = \langle S, s^0, \delta, Acc\rangle$ be that deterministic automaton, where Acc is a parity condition using a function $\mathsf{c} : S \to \mathbb{N}$. The finite tree with back edges $\overline{\tau} = \langle \overline{T}, \overline{\mathcal{L}}\rangle$, where $\overline{\mathcal{L}} \subseteq 2^{sub(\varphi)} \times \mathcal{R}eg \times S$, is constructed from τ as follows:

- if m is a root of τ, $\mathcal{L}(m) = \{\varphi\}; r$, and $\delta(s^0, (\{\varphi\}; r)) = s$ then m added to nodes of $\overline{\tau}$ and set $\overline{\mathcal{L}}(m) = \mathcal{L}(m) \times \{s\}$.
- if m is a node of $\overline{\tau}$ with $\overline{\mathcal{L}}(m) = (\Gamma; r, s)$, and n is a son of the unique correspondent of m in τ, then n is added in $\overline{\tau}$ as a son of m and, $\overline{\mathcal{L}}(n) = \mathcal{L}(n) \times \{s'\}$ with $s' = \delta(s, \mathcal{L}(n))$. An exception to this occurs when there is an ancestor n' of m in $\overline{\tau}$ with $\mathcal{L}(n') = \mathcal{L}(n)$ and $\mathsf{c}(\mathcal{L}_S(n)) = \max\{\mathsf{c}(m') \mid m' \text{ occurs between } n' \text{ and } m\}$. In this case, a back edge from m to n' is added in τ. If $\mathsf{c}(\mathcal{L}_S(n'))$ is even, then assign the color *red* to n', else assign the color green.

By construction, $\overline{\tau}$ unwinds to τ. Consider an infinite path $\pi = n_1, n_2, n_3, \ldots$ of the unwinding of $\overline{\tau}$; this path has a unique correspondent in $\overline{\tau}$ and therefore it contains either a μ-trace or a ν-trace. Moreover, there is i and j such that $n_i = n_j$ and n_i is the highest node from which the type of the path is decided. By construction we get that n_i is of color red if and only if the path π contains a μ-trace. □

End of the proof of Theorem 113 Here, we end the proof of Theorem 113 and we construct the equivalent disjunctive normal form formula to a given formula. The idea is to use the tableau with back edges defined above to get the formula in the desired form. For that purpose, a formula will be assigned to each node of tableau with back edges depending on the rule applied in the node. This should be clear for all nodes, except nodes with a back edge and nodes at which the rule (*mod*) is applied. Let us go into the proof that shows the construction of an equivalent disjunctive normal form formula.

Recall that a tableau with back edges does not have an infinite path. A leaf of such a tableau contains either tt, ff or formulas of the form $(\hat{g}, a) \rightarrow\emptyset$. We start the construction of the disjunctive normal form formula from the leaf of the tableau with back edge and we move to its root by assigning a disjunctive formula $\breve{\varphi}_n$ to each node of the tableau with back edge in the following way:

1. If n is a leaf then $\breve{\varphi}_n$ is the conjunction of all the literals and formulas of the form $(\hat{g}, a) \rightarrow\emptyset$ in $\overline{\mathcal{L}}_{ERL}(n)$.
2. In the case that there are outgoing edges from n, we assume that every son of n has assigned some disjunctive formula. We also assume that a formula assigned to a son is assigned to an edge leading from n to this son. We assign the variable X_m to a back edge, if this back edge leads to an ancestor m. An auxiliary formula γ_n is assigned to each internal node n according to the rule applied in n. This assignment works as follows:

- if one of the rules $(\wedge)$, (reg), (ν), or (μ) was applied in n, and γ' is the formula on the unique edge leading from n, then an *auxiliary* formula $\gamma_n = \gamma'$ is assigned to n.
- if the rule $(\vee)$ was applied in n, then the *auxiliary* formula $\gamma_n = \breve{\varphi}_{n_1} \vee \breve{\varphi}_{n_2}$ is assigned to n where $\breve{\varphi}_{n_1}$ and $\breve{\varphi}_{n_2}$ are the disjunctive formulas assigned to each edge leading from n;
- if the rule (mod) was applied in n then let $\Gamma_{\hat{g},a}$ be the set of all the formulas assigned to edges leading from n to some node labeled by a result of reduction of the guarded event $(\hat{g}, a)$. We let γ_n be the conjunction of all the literals and formulas of the form $(\hat{g}, a) \rightarrow \emptyset$ appearing in $\overline{\mathcal{L}}(n)$ together with all the formulas of the form $(\hat{g}, a) \rightarrow \Gamma_{\hat{g},a}$.

In the case that there is no back edge leading to n, then $\breve{\varphi}_n = \gamma_n$. Otherwise $\breve{\varphi}_n = \mu X_n.\gamma_n$ if n is colored red and, $\breve{\varphi}_n = \nu X_n.\gamma_n$ is colored green.

To end the proof of the theorem 113, we claim that, using the construction of $\breve{\varphi}$, is easy to construct a tableau $\tau^{\breve{\varphi}}_{r^0}$ for $\breve{\varphi}$ and a function $\mathcal{E} : \tau^{\breve{\varphi}}_{r^0} \rightarrow \tau^{\varphi}_{r^0}$ that defines an equivalence between $\tau^{\breve{\varphi}}_{r^0}$ and $\tau^{\varphi}_{r^0}$. In consequence $\breve{\varphi}$ and φ are equivalent.

3.6 Concluding Remarks

We have considered Event-Recording Logic as a language for describing properties of timed processes. We have presented an algorithm for the model-checking problem of ERL formulas. The algorithm uses the M-region representation of timed processes and that is similar to a well known algorithm for the model-checking problem of the standard μ-calculus. We wondered, if other results in the setting of the standard μ-calculus could be transferred to the setting of Event-Recording Logic. We have shown that the answer is yes for the satisfiability, and the disjunctive normal form property. From the model-checking algorithm, we have provided a new tableau system for the satisfiability checking problem of Event-Recording Logic. Our tableau system of rules is different and simpler than the tableau system proposed earlier by Sorea. Then we have pointed out some ambiguities when using tableau system of Sorea and some incorrectness caused by the use of approximation operation on zones. The simplicity of our tableau system of rules has enabled us to provide a disjunctive normal form theorem for ERL formulas.

Chapter 4

The Logic WT_μ

An important modality over occurrences of an event in a real-time system is the *the necessity modality* on the time periods at which the event can occur. The necessity modality allows to describe general properties like *"An event can be completed at every time instance when a condition on the time is satisfied"*. Examples of such kinds of properties are: *"After a coin is inserted, coffee is continuously available for 30 seconds"* or *"the brake system of a car operates at any time within the 10 time units"*. We claim that the modalities of ERL can not be used to handle such important kinds of properties.

In this chapter, we introduce a new logic that we call WT_μ. The logic WT_μ is a weak timed extension of the standard μ-calculus. Formulas of WT_μ are interpreted over timed processes. Its modalities are indexed with either constraints or events, while modalities of ERL are indexed with pairs made of a constraint and an event. We show that WT_μ is more expressive than ERL as every formula of ERL can be translated into an equivalent WT_μ formula; and there are some formulas of WT_μ that can not be translated into formulas of ERL. Modalities of WT_μ are of the form $\langle g \rangle$ and $[g]$ in addition to the classical modalities of the μ-calculus indexed with event ($\langle a \rangle$ and $[a]$). Intuitively, a state of a timed process p satisfies $\langle g \rangle \varphi$ from a given time-context with a valuation v if by letting the time elapse in it, it is possible to reach a moment when the values of the clocks satisfy g and in that moment, the formula φ is satisfied. A state p of a timed process satisfies $[g]\varphi$ from a time-context v if whenever starting from v we let the time pass and reach a moment when g is satisfied then φ is satisfied in that moment. We consider the model-checking and the satisfiability problems for WT_μ as they can be then used for the controller synthesis.

For the model-checking problem, we use our approach to the model-checking of ERL, so we reduce the model-checking problem of WT_μ to the model-checking problem of the standard μ-calculus.

For the satisfiability, we will consider fragments of WT_μ as the satisfiability of WT_μ itself is difficult. We consider a first fragment that we call WG-WT_μ (for Well Guarded WT_μ) and a second fragment that we call C-WT_μ (for WT_μ for controller synthesis). Roughly speaking, Formulas WG-WT_μ are formulas of WT_μ such that every modality indexed with a constraint is immediately followed by a boolean combination of formulas all starting with a modality indexed with an event; and a modality indexed with an event is preceded by a modality indexed with a constraint. Formulas of C-WT_μ disallow an existential modality

indexed with a constraint ($\langle g \rangle$) to be followed with a combination of formulas containing a formula starting with a universal modality indexed with an event ($[a]$) except $[a]tt$. C-WT_μ is a fragment of WG-WT_μ. We provide a tableau system of rules for C-WT_μ and we show, by using techniques similar to the ones in the previous chapter, that the satisfiability problem for C-WT_μ is decidable. Then, we wonder whether we could use our tableau system of rules for deciding whether a C-WT_μ formula has a deterministic model. We show that this problem is not easy as it could involve the use of new integer constants in the models.

Related results: Logics (TML [HLY91], L^t_μ [SS95] L_ν [LLW95]) that enable to describe the *the necessity modal operator* has been considered for describing properties of timed automata but the decidability of the satisfiability problem has not been established. Laroussinie et al. [LLW95] have introduced the logic L_ν as a more powerful logic than the one in [HLY91, SS95] but its satisfiability problem is still open and no disjunctive normal form has been provided [BCL05]. The logics L_ν and WT_μ are incomparable as they are not interpreted over the same model and L_ν does not allow the least fixpoint operator. But, if we restrict the interpretation of L_ν to timed processes, we get that $\langle g \rangle \varphi$ has the same meaning as the L_ν formula $\langle \delta \rangle (g \wedge \varphi)$, $[g]\varphi$ has the same meaning as the L_ν formula $[\delta](g \rightarrow \varphi)$, $\langle a \rangle \varphi$ has the same meaning as $\langle a \rangle (x_a \ \underline{in} \ \varphi)$ and $[a]\varphi$ has the same meaning as $[a](x_a \ \underline{in} \ \varphi)$.

This chapter is organised as follows: We define WT_μ, WG-WT_μ and C-WT_μ in the next section. In Section 4.2 we consider the model-checking of WT_μ and we present the relation between WT_μ and ERL. In Section 4.3 we use tableau-based technique to show the decidability of the satisfiability problem for C-WT_μ.

4.1 Syntax and Semantics

We define the syntax of WT_μ, WG-WT_μ and C-WT_μ formulas. WT_μ formulas have modalities indexed with constraints and modalities indexed with events. We define rectangular formulas that use only rectangular constraints and we show that every formula can be transformed into an M-equivalent rectangular formula. Then we show that modalities of ERL can be simulated by combinations of modalities of WT_μ, meaning that ERL is a fragment of WT_μ.

4.1.1 Definitions

The logic WT_μ is a variant of the μ-calculus and ERL. The formulas of WT_μ describe properties on timed processes. Apart from the usual events modalities of the standard μ-calculus, it has also modalities indexed by constraints. Modalities of WT_μ can also be seen as an adaptation of modalities of L_ν for timed processes.

Definition 119 Let X, Y range over the set of variables denoted Var. A formula φ of WT_μ is generated using the following grammar:

$$\varphi ::= tt \mid ff \mid X \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle a \rangle \varphi \mid \langle g \rangle \varphi \mid [a]\varphi \mid [g]\varphi \mid \mu X.\varphi \mid \nu X.\varphi$$

where $a \in \Sigma$ is an event and $g \in Gds_\Sigma$ is a constraint.

The *bound* of a formula is the maximal constant that occurs in its constraints. For a formula φ, M_φ denotes its bound. Given a constant M, we say that a formula is M-bounded if its bound is smaller than M.

Notion of bound variables, sentences, sub formulas, well named formula, ν-variable, μ-variable, dependency order, alternation depth, guarded formulas, expansion, and binding function are analogous to the definitions of similar notions for the setting of the μ-calculus in Section 1.3.

4.1.2 Semantics of WT_μ

A formula is interpreted over timed processes, or rather their semantics. Intuitively, we say that a state (p, v) satisfies a formula $[g]\varphi$, if whenever starting from v we let the time pass and reach a valuation $v' \vDash g$ then $(p, v') \vDash_t \varphi$. Similarly, a formula $\langle g\rangle\varphi$ is satisfied if by letting the time pass it is possible to go from valuation v to a valuation $v' \vDash g$ with $(p, v') \vDash_t \varphi$. The meaning for the modalities $[a]$ and $\langle a\rangle$ is classical.

We will be mainly interested in describing timed processes, but actually the formulas of WT_μ can be evaluated in any $(\mathcal{V}_\Sigma \cup \Sigma)$- labelled transition system. Let us fix such a system $\mathcal{S} = \langle S, \Sigma \cup \mathcal{V}_\Sigma, s^0, \Delta_\mathcal{S}\rangle$. The semantics of a formula φ, denoted $[\![\varphi]\!]^\mathcal{S}_{Val}$, defined with respect to an assignment $Val : Var \to 2^S$ is the set of states of $\mathcal{S}$ which satisfy φ.

We write $\mathcal{S}, s, Val \vDash_t \varphi$ to say that the state s satisfies φ with respect to the valuation Val.

Definition 120 For a given $(\mathcal{V}_\Sigma \cup \Sigma)$-labelled transition system $\mathcal{S}$, a given formula φ and an assignment $Val : Var \to \mathcal{P}(S)$, we define the satisfaction relation $\vDash_t$ and the semantics $[\![\varphi]\!]^\mathcal{S}_{Val}$ inductively as follows:

- $[\![\varphi]\!]^\mathcal{S}_{Val} = \{s \mid \mathcal{S}, s, Val \vDash_t \varphi\}$
- $\mathcal{S}, s, Val \vDash_t tt$.
- $\mathcal{S}, s, Val \vDash_t X$ if $s \in Val(X)$.
- $\mathcal{S}, s, Val \vDash_t \varphi_1 \vee \varphi_2$ if $\mathcal{S}, s, Val \vDash_t \varphi_1$ or $\mathcal{S}, s, Val \vDash_t \varphi_2$.
- $\mathcal{S}, s, Val \vDash_t \varphi_1 \wedge \varphi_2$ if $\mathcal{S}, s, Val \vDash_t \varphi_1$ and $\mathcal{S}, s, Val \vDash_t \varphi_2$.
- $\mathcal{S}, s, Val \vDash_t \langle a\rangle\varphi$ if there is $s \xrightarrow{a} s'$ such that $\mathcal{S}, s', Val \vDash_t \varphi$.
- $\mathcal{S}, s, Val \vDash_t \langle g\rangle\psi$ if there is $s \xrightarrow{v} s'$ such that $v \in [\![g]\!]$ and $\mathcal{S}, s', Val \vDash_t \psi$.
- $\mathcal{S}, s, Val \vDash_t [a]\varphi$ if for all $s \xrightarrow{a} s'$ we have $\mathcal{S}, s', Val \vDash_t \varphi$.
- $\mathcal{S}, s, Val \vDash_t [g]\psi$ if for all $s \xrightarrow{v} s'$ with $v \in [\![g]\!]$, we have $\mathcal{S}, s', Val \vDash_t \psi$.
- $\mathcal{S}, s, Val \vDash_t \mu X.\varphi(X)$ if $s \in \bigcap\{T \subseteq S \mid [\![\varphi(X)]\!]^\mathcal{S}_{Val[X/T]} \subseteq T\}$.
- $\mathcal{S}, s, Val \vDash_t \nu X.\varphi(X)$ if $s \in \bigcup\{T \subseteq S \mid T \subseteq [\![\varphi(X)]\!]^\mathcal{S}_{Val[X/T]}\}$.

We will write $\mathcal{S} \vDash_t \varphi$ for $\mathcal{S}, s^0 \vDash_t \varphi$ to say that $\mathcal{S}$ is a *model* of the sentence φ.

To ensure the existence of fixpoints, we need to show that modal operators indexed with constraints and modal operators indexed with events are monotone.

Proposition 121 The operators $\langle\alpha\rangle$ and $[\alpha]$ are monotone for every $\alpha \in Gds_\Sigma \cup \Sigma$.

Proof
The cases for operators other than $\langle g\rangle$ and $[g]$ are standard. We show that modal operators indexed with constraints are monotonic. Assume that we have φ_1 and φ_2 and a transition system $\mathcal{S}$ such that $[\![\varphi_1]\!]^{\mathcal{S}}_{Val} \subseteq [\![\varphi_2]\!]^{\mathcal{S}}_{Val}$

- If $s \in [\![\langle g\rangle\varphi_1]\!]^{\mathcal{S}}_{Val}$ then there is $s \xrightarrow{v} s'$ with $v \in [\![g]\!]$ such that $s' \in [\![\varphi_1]\!]^{\mathcal{S}}_{Val}$ and then $s \in [\![\varphi_2]\!]^{\mathcal{S}}_{Val}$ as $[\![\varphi_1]\!]^{\mathcal{S}}_{Val} \subseteq [\![\varphi_2]\!]^{\mathcal{S}}_{Val}$. Then $s \in [\![\langle g\rangle\varphi_2]\!]^{\mathcal{S}}_{Val}$
- If $s \in [\![[g]\varphi_1]\!]^{\mathcal{S}}_{Val}$ and $s \notin [\![[g]\varphi_2]\!]^{\mathcal{S}}_{Val}$ then there is $s \xrightarrow{v} s'$ with $v \in [\![g]\!]$ such that $s' \notin [\![\varphi_2]\!]^{\mathcal{S}}_{Val}$. As $s' \notin [\![\varphi_2]\!]^{\mathcal{S}}_{Val}$ and $[\![\varphi_1]\!]^{\mathcal{S}}_{Val} \subseteq [\![\varphi_2]\!]^{\mathcal{S}}_{Val}$ we get that $s' \notin [\![\varphi_1]\!]^{\mathcal{S}}_{Val}$. Then there is $s \xrightarrow{v} s'$ with $v \in [\![g]\!]$ such that $s' \notin [\![\varphi_1]\!]^{\mathcal{S}}_{Val}$. So $s \notin [\![[g]\varphi_1]\!]^{\mathcal{S}}_{Val}$, a contradiction.

□

We write $\varphi_1 \equiv \varphi_2$ when the formulas φ_1 and φ_2 are equivalent.

We introduce the negation operator $\neg$. Given a sentence φ, a $(\mathcal{V} \times \Sigma)$-labelled transition system $\mathcal{S}$, and a valuation Val, we define $[\![\neg\varphi]\!]^{\mathcal{S}}_{Val} = S \setminus [\![\varphi]\!]^{\mathcal{S}}_{Val}$

Proposition 122 We have the following equivalences.

1. $\neg tt \equiv ff$
2. $\neg ff \equiv tt$
3. $\neg(\varphi_1 \wedge \varphi_2) \equiv \neg\varphi_1 \vee \neg\varphi_2$
4. $\neg(\varphi_1 \vee \varphi_2) \equiv \neg\varphi_1 \wedge \neg\varphi_2$
5. $\neg\langle\alpha\rangle\varphi \equiv [\alpha]\neg\varphi \quad$ for $\alpha \in \Sigma \cup Gds$
6. $\neg[\alpha]\varphi \equiv \langle\alpha\rangle\neg\varphi \quad$ for $\alpha \in \Sigma \cup Gds$
7. $\neg\mu X.\varphi(X) \equiv \nu X.\neg\varphi(\neg X)$
8. $\neg\nu X.\varphi(X) \equiv \mu X.\neg\varphi(\neg X)$

Proof
Let $\mathcal{S}$ be a $(\Sigma \cup \mathcal{V}_\Sigma)$-labelled transition system and let s be a state of $\mathcal{S}$. As the cases for operators other than $\langle g\rangle$ and $[g]$ are standard, we consider the following:

- If $s \in [\![\neg\langle g\rangle\varphi]\!]$ then $s \notin [\![\langle g\rangle\varphi]\!]$. It is equivalent to say that for every $v \in [\![g]\!]$, for every $s \xrightarrow{v} s'$, we have that $s' \notin [\![\varphi]\!]$ meaning by definition that $s' \in [\![[g]\neg\varphi]\!]$.
- The case of $\neg[g]\varphi \equiv \langle g\rangle\neg\varphi$ is obvious from the previous case.

□

Proposition 123 Let $g, g_1, g_2, \ldots, g_n$ such that $[\![g]\!] = \bigcup_{i=1..n} [\![g_i]\!]$ then,

1. $\langle g\rangle\varphi \equiv \bigvee_{i=1..n}\langle g_i\rangle\varphi$
2. $[g]\varphi \equiv \bigwedge_{i=1..n}[g_i]\varphi$

Proof
We will consider the first case since the proof of the second case is easy by using Proposition 122. Let $\mathcal{S}$ be a $(\Sigma \cup \mathcal{V}_\Sigma)$-labelled transition system and s be a state of $\mathcal{S}$

($\Longrightarrow$) If $\mathcal{S}, s \vDash_t \langle g\rangle\varphi$ then there is $s \xrightarrow{v} s'$ with $v \in [\![g]\!]$ such that $\mathcal{S}, s' \vDash_t \varphi$. As $[\![g]\!] = \bigcup_{i=1..n}[\![g_i]\!]$, there is $i \in [1..n]$ such that $v \in [\![g_i]\!]$. Then, $s \xrightarrow{v} s'$ with $v \in [\![g_i]\!]$ and $\mathcal{S}, s' \vDash_t \varphi$, meaning that $\mathcal{S}, s \vDash_t \langle g_i\rangle\varphi$ hence $\mathcal{S}, s \vDash_t \bigvee_{i=1..n}\langle g_i\rangle\varphi$.

($\Longleftarrow$) If $\mathcal{S}, s \vDash_t \bigvee_{i=1..n}\langle g_i\rangle\varphi$ then $\mathcal{S}, s \vDash_t \langle g_i\rangle\varphi$ for some $i \in [1..n]$ meaning that, there is $s \xrightarrow{v} s'$ with $v \in [\![g_i]\!]$ such that $\mathcal{S}, s' \vDash_t \varphi$. But $v \in [\![g_i]\!]$ implies $v \in [\![g]\!]$ as $[\![g]\!] = \bigcup_{i=1..n}[\![g_i]\!]$. Then we get that $\mathcal{S}, s \vDash_t \langle g\rangle\varphi$.

□

Meaning of a formula over a timed process Consider φ a formula, and $\mathcal{P}$ a timed process. We say that φ is satisfied in a state p, a valuation of clocks $v : H \to \mathbb{R}$, and a valuation $Val : Var \to \mathcal{P}(P \times \mathcal{V}_\Sigma)$ of propositional variables and we write $\mathcal{P}, (p, v), Val \vDash \varphi$ when $[\![\mathcal{P}]\!], (p, v), Val \vDash_t \varphi$.

The meaning $[\![\varphi]\!]^{\mathcal{P}}_{Val} \subseteq P \times \mathcal{V}_\Sigma$ of a formula over a timed process $\mathcal{P}$ is defined by

$$[\![\varphi]\!]^{\mathcal{P}}_{Val} = [\![\varphi]\!]^{[\![\mathcal{P}]\!]}_{Val}$$

We will write $\mathcal{P} \vDash \varphi$ if $[\![\mathcal{P}]\!]$ is a model of φ and we say that $\mathcal{P}$ is a *model* of φ.

4.1.3 Restricted Logics: WG-WT$_\mu$ and C-WT$_\mu$

We will consider fragments of WT$_\mu$. The first fragment of WT$_\mu$ we will consider is WG-WT$_\mu$ and the second one is WT$_\mu$ for the control (C-WT$_\mu$). C-WT$_\mu$ is itself a fragment of WG-WT$_\mu$.

Definition 124 The set of WG-WT$_\mu$ formulas is defined by the following rules:

- tt, ff and X are formulas of WG-WT$_\mu$.
- $\langle g\rangle\varphi$ and $[g]\varphi$ are formulas of WG-WT$_\mu$ provided that φ is a boolean combination of formulas of the form $\langle a\rangle\psi$ or $[a]\psi$ where ψ is a formula of WG-WT$_\mu$
- $\varphi \wedge \psi$ and $\varphi \vee \psi$ are formulas of WG-WT$_\mu$ provided that φ and ψ are formulas of WG-WT$_\mu$.
- $\mu X.\varphi$ and $\nu X.\varphi$ are formulas of WG-WT$_\mu$ provided that φ is a formula of WG-WT$_\mu$.

We remark that a formula of WG-WT$_\mu$ is also a formula of WT$_\mu$. Formulas of WG-WT$_\mu$ are such that if we look at a formula as a tree, then the modalities indexed with constraints and with events must alternate on each path.

We also remark that $\langle g\rangle ff$ is equivalent to $\langle g\rangle \bigvee_{a\in\Sigma}\langle a\rangle ff$ and $\langle g\rangle tt$ is equivalent to $\langle g\rangle \bigwedge_{a\in\Sigma}[a]tt$. By the duality principle, $[g]ff$ is equivalent to $[g]\bigvee_{a\in\Sigma}\langle a\rangle ff$ and $[g]tt$ is equivalent to $[g]\bigwedge_{a\in\Sigma}\langle a\rangle tt$. Then we can allow modalities indexed with constraints to be followed by tt and ff without changing the definition of WG-WT$_\mu$ syntax.

Definition 125 The set of C-WT$_\mu$ formulas are defined by the following rules:

- tt, ff and X are formulas of C-WT$_\mu$.
- $\langle g\rangle\varphi$ is a formula of C-WT$_\mu$ provided that φ is a positive boolean combination of formulas of the form $\langle a\rangle\psi$ where ψ is a formula of C-WT$_\mu$.
- $[g]\varphi$ are formulas of C-WT$_\mu$ provided that φ is a boolean combination of formulas of the forms $\langle a\rangle\psi$ or $[a]\psi$ where ψ is a formula of C-WT$_\mu$.
- $\varphi\wedge\psi$ and $\varphi\vee\psi$ are formulas of C-WT$_\mu$ provided that φ and ψ are formulas of C-WT$_\mu$.
- $\mu X.\varphi$ and $\nu X.\varphi$ are formulas of C-WT$_\mu$ provided that φ is a formula of C-WT$_\mu$.

By definition C-WT$_\mu$ is a fragment of WG-WT$_\mu$. Indeed, in formulas of C-WT$_\mu$ a formula of the form $[a]\varphi$ is not allowed after an existential delay modality. We remark that since $tt\equiv[a]tt$, we can allow formulas of the form $[a]tt$ to occur in the set of formulas participating in the boolean combination that follows an existential delay modality indexed with a constraint; this does not change the expressive power of C-WT$_\mu$.

Example: In the WG-WT$_\mu$ formula $\varphi=\langle 0<h_a<1\rangle((\langle b\rangle tt\wedge[a]ff)\vee\langle c\rangle tt)$ events a, b and c are in the scope of the modality $\langle 0<h_a<1\rangle$. The formula φ says that there is a time at which $0<h_a<1$ is satisfied and at that time, the event c can be completed or the event b can be completed and the event a can not be completed. We observe that φ is not a formula of C-WT$_\mu$. □

4.1.4 Rectangular Formulas

We introduce rectangular form for WT$_\mu$ formulas and we show the equivalence between a formula and its rectangular form.

Definition 126 A *rectangular formula* is a formula defined using rectangular constraints.

Recall that $Rect_M(g)$ was presented in Definition 56. The M-rectangular formula associated to the formula φ is the formula $Rect_M(\varphi)$ inductively defined by:

- $Rect_M(ff)=ff$
- $Rect_M(tt)=tt$
- $Rect_M(X)=X$

- $Rect_M(\varphi_1 \wedge \varphi_2) = Rect_M(\varphi_1) \wedge Rect_M(\varphi_2)$
- $Rect_M(\varphi_1 \vee \varphi_2) = Rect_M(\varphi_1) \vee Rect_M(\varphi_2)$
- $Rect_M(\langle g \rangle \varphi) = \bigvee_{\hat{g} \in Rect_M(g)} \langle \hat{g} \rangle \varphi$
- $Rect_M([g]\varphi) = \bigwedge_{\hat{g} \in Rect_M(g)} [\hat{g}]\varphi$
- $Rect_M(\langle a \rangle \varphi) = \langle a \rangle Rect_M(\varphi)$
- $Rect_M([a]\varphi) = [a] Rect_M(\varphi)$
- $Rect_M(\sigma X.\varphi(X)) = \sigma X.Rect_M(\varphi(X))$ where σ is one of $\{\mu, \nu\}$

We can state the following proposition.

Proposition 127 For every $M \geq M_\varphi$, $\mathcal{S}, s, Val \models_t \varphi$ if and only if $\mathcal{S}, s, Val \models_t Rect_M(\varphi)$

Proof
The proof uses structural induction.

- The cases of *ff*, *tt*, X are standard.
- The cases of formulas of the form $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$ are also standard.
- If $\mathcal{S}, s, Val \models_t \langle a \rangle \varphi$, then there is $s \xrightarrow{a} s'$ with $v' = v[h_a := 0]$ such that $\mathcal{S}, s' \models_t \varphi$. By induction hypothesis, $\mathcal{S}, s' \models_t Rect_M(\varphi)$. It follows that $\mathcal{S}, s, Val \models_t Rect_M(\langle a \rangle \varphi)$. The other way of the proof uses similar argumentation.
- The case of $[a]\varphi$ uses dual argumentation.
- The case when $\varphi = \langle g \rangle \varphi$. $Rect_M(\varphi) = \bigvee_{\hat{g} \in Rect_M(g)} \langle \hat{g} \rangle \varphi$. From Proposition 57, $[\![g]\!] = \bigcup_{\hat{g} \in Rect_M(g)} [\![\hat{g}]\!]$. We use Proposition 123 to conclude.
- Argumentation for the case when $\varphi = [g]\varphi$ is similar to the case when $\varphi = \langle g \rangle \varphi$.
- The cases of fixpoint formulas are standard.

□

4.1.5 Relation between ERL and WT_μ

We show that ERL is a fragment of WT_μ. With an example, we show that modal operators we have introduced are useful for describing some relevant real-time properties on timed processes in particular the necessity modal property on time delay.

Proposition 128 Consider a property that can be written using a WT_μ formula φ or an ERL formula ψ, then for every timed process $\mathcal{P}$, state p of $\mathcal{P}$ and valuation $v \in \mathcal{V}_\Sigma$,

- $\mathcal{P}, (p, v), Val \models_t \langle g \rangle \langle a \rangle \varphi$ if and only if $\mathcal{P}, (p, v), Val \models_t \langle g, a \rangle \psi$.

- $\mathcal{P}, (p, v)$, $Val \models_t [g][a]\varphi$ if and only if $\mathcal{P}, (p, v)$, $Val \models_t [g, a]\psi$.

Lemma 129 There is a property that can be described with a WT_μ formula and that can not be described with an ERL formula.

Proof
Consider the property *"in the time interval* $(0, 1)$ *there is a time instance when no action a is possible"*.. This property can be expressed by WT_μ formula:

$$\varphi = \langle 0 \leq h_a < 1\rangle[a]ff$$

Observe that we use the clock associated to action a, but we could use any other clock as we assume that initial valuation of all clocks is 0. Of course φ is satisfiable, moreover it is consistent with the formula

$$\varphi' = \langle 0 \leq h_a < 1\rangle\langle a\rangle tt$$

saying that there is a time instance when a is possible. We show that φ is not equivalent to a ERL formula. We claim that any ERL formula consistent with φ' is not equivalent to φ. Indeed, every ERL formula can be transformed into a boolean combination of formulas starting with modalities $\langle g, b\rangle$ or $[g, b]$. It is easy to verify that every such formula that is consistent with φ' has a model where action a is possible at every time instance between 0 and 1. □

In consequence of Lemma 129 and Proposition 128 we get the following.

Theorem 130 *WT_μ is strictly more expressive than ERL*

Example: Assume that we aim at checking the following property of timed process in Figure 15.

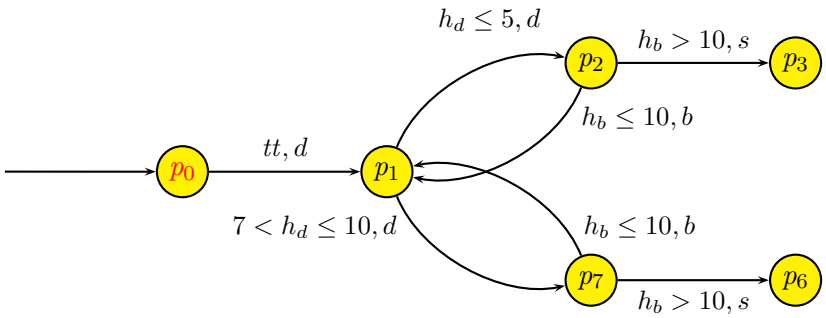

Figure 15: A toy-car model.

The system operates at any time within the 10 time units after the first ***d*** *signal by sending a second* ***d*** *signal; it should send signal* ***s*** *at least 10 time units after the second* ***d*** *or receive signal* ***b*** *at most 10 time units after the second* ***d***.
This property is described with the following WT_μ formula:

$$\varphi ::= [tt][d][h_d \leq 10]\langle d\rangle(\langle h_b \leq 10\rangle\langle b\rangle tt \vee \langle h_d > 10\rangle\langle s\rangle tt)$$

The system modeled in Figure 15 is not a model of φ. For example, if the second "danger signal" occurs 6 time units after the first "danger signal" the system will never compute the following "brake signal" unless another "danger signal" occurs 2 time units after the second. So there is a risk that the car goes into collision. $\square$

4.2 Model-checking

We consider the model-checking of WT_μ. We define the abstract semantics of formulas in which formulas are interpreted over $(Gds_\Sigma \cup \Sigma)$-labelled transition systems. In that semantics constraints in transitions are directly compared (identity test) with the constraints in formulas. Then we use that semantics for the model-checking by showing that checking if a timed process is a model of a formula is the same as checking if the M-region semantics of that timed process is an abstract model (with respect to the abstract semantics) of the M-rectangular formula of the formula for M sufficiently big.

4.2.1 Abstract Semantics for Formulas

We would also like to evaluate our formulas in models of the form $\langle\!\langle \mathcal{P} \rangle\!\rangle$ or $\langle\!\langle \mathcal{P} \rangle\!\rangle^M_{reg}$. More generally, we can define a semantics of WT_μ in any $(Gds_\Sigma \cup \Sigma)$-labelled transition system $\mathcal{S} = \langle S, Gds_\Sigma \cup \Sigma, s^0, \rightarrow \rangle$ as follows:

Definition 131 The *symbolic relation of satisfaction*, $\mathcal{S}, s, Val \vDash_g$, and symbolic meaning of a formula ${}^g[\![\varphi]\!]^{\mathcal{S}}_{Val}$ are defined for a symbolic representation $\mathcal{S}$, a valuation of variables Val and a formula φ inductively as follows:

- $\mathcal{S}, s, Val \vDash_g tt$
- $\mathcal{S}, s, Val \vDash_g X$ when $s \in Val(X)$
- $\mathcal{S}, s, Val \vDash_g \varphi_1 \vee \varphi_2$ when $\mathcal{S}, s, Val \vDash_g \varphi_1$ or $\mathcal{S}, s, Val \vDash_g \varphi_2$.
- $\mathcal{S}, s, Val \vDash_g \varphi_1 \wedge \varphi_2$ when $\mathcal{S}, s, Val \vDash_g \varphi_1$ and $\mathcal{S}, s, Val \vDash_g \varphi_2$.
- $\mathcal{S}, s, Val \vDash_g \langle a \rangle \psi$ if there is $s \xrightarrow{a} s'$ such that $\mathcal{S}, s', Val \vDash_g \psi$
- $\mathcal{S}, s, Val \vDash_g \langle g \rangle \psi$ if there is $s \xrightarrow{g} s'$ such that $\mathcal{S}, s', Val \vDash_g \psi$
- $\mathcal{S}, s, Val \vDash_g [a] \psi$ if for all $s \xrightarrow{a} s'$ we have $\mathcal{S}, s', Val \vDash_g \psi$
- $\mathcal{S}, s, Val \vDash_g [g] \psi$ if for all $s \xrightarrow{g} s'$ we have $\mathcal{S}, s', Val \vDash_g \psi$
- $\mathcal{S}, s, Val \vDash_g \mu X.\varphi(X)$ if $s \in \bigcap \{T \subseteq S \mid \langle\!\langle \varphi(X) \rangle\!\rangle^{\mathcal{S}}_{Val[X/T]} \subseteq T\}$
- $\mathcal{S}, s, Val \vDash_g \nu X.\varphi(X)$ if $s \in \bigcup \{T \subseteq S \mid T \subseteq \langle\!\langle \varphi(X) \rangle\!\rangle^{\mathcal{S}}_{Val[X/T]}\}$
- ${}^g[\![\varphi]\!]^{\mathcal{S}}_{Val} = \{s \mid \mathcal{S}, s, Val \vDash_g \varphi\}$

We will write $\mathcal{S} \models_g \varphi$ for $\mathcal{S}, s^0 \models_g \varphi$ to say that $\mathcal{S}$ is an *abstract model* of the sentence φ.

Observe that this is nothing else, but the semantics of the standard μ-calculus. We use this observation in the next subsection for the model-checking decision procedure. Results we present in that subsection use the framework of Subsection 3.2.3. Constructs for approximating fixpoints in WT_μ formulas are analogous to the ones in Subsection 3.2.2.

4.2.2 Model-Checking Results

Let us now consider the model-checking of WT_μ. From Proposition 127, we can consider rectangular formula as "good" abstraction of formula and for sufficiently big M, we will use the M-region representation of timed process $\mathcal{P}$, to check whether it is a model of a given formula.

Proposition 132 For every process $\mathcal{P}$, for every M_φ-rectangular formula φ, for every $M \geq M_\varphi$: $[\![\mathcal{P}]\!], (p, v), \mathit{Val} \models_t \varphi$ if and only if $\langle\!|\mathcal{P}|\!\rangle^M, (p, v), \mathit{Val} \models_g \varphi$.

Proof
The proof is by induction on the structure of the formula. The cases of ff, tt, $\varphi \vee \varphi$, $\varphi \wedge \varphi$ and $\sigma X.\varphi(X)$ are immediate. We consider the cases of $\langle g \rangle \varphi$, $[g]\varphi$, $\langle a \rangle \varphi$ and $[a]\varphi$.

- Assume that the formula has the form $\langle g \rangle \varphi$ where, $g \in \mathit{Agds}(M)$.

  $\Rightarrow$ if $[\![\mathcal{P}]\!], (p, v), \mathit{Val} \models_t \langle g \rangle \varphi$, then there is $(p, v) \xrightarrow{v'} (p, v')$ such that $v' \in [\![g]\!]$ and $[\![\mathcal{P}]\!], (p, v'), \mathit{Val} \models_t \varphi$. By the induction hypothesis, $\langle\!|\mathcal{P}|\!\rangle^M, (p, v'), \mathit{Val} \models_g \varphi$. But, $(p, v) \xrightarrow{v'} (p, v')$, $v' \in [\![g]\!]$ and $g \in \mathit{Agds}(M)$ involve that $(p, v) \xrightarrow{g} (p, v')$ is a transition in $\langle\!|\mathcal{P}|\!\rangle^M$. It follows that $\langle\!|\mathcal{P}|\!\rangle^M, (p, v), \mathit{Val} \models_g \langle g \rangle \varphi$.

  $\Leftarrow$ $\langle\!|\mathcal{P}|\!\rangle^M, (p, v), \mathit{Val} \models_g \langle g \rangle \varphi$, then there is $(p, v) \xrightarrow{g} (p, v')$ such that $[\![\mathcal{P}]\!], (p, v'), \mathit{Val} \models_g \varphi$. By the induction hypothesis, $[\![\mathcal{P}]\!], (p, v'), \mathit{Val} \models_t \varphi$. But if $(p, v) \xrightarrow{g} (p, v')$ is a transition in $\langle\!|\mathcal{P}|\!\rangle^M$ then $v' \in [\![g]\!]$ and there is $t \in \mathbb{R}^+$ such that $v' = v + t$. It follows that, the transition $(p, v) \xrightarrow{v'} (p, v')$ belong to $[\![\mathcal{P}]\!]$ and then $[\![\mathcal{P}]\!], (p, v), \mathit{Val} \models_t \langle g \rangle \varphi$.

- In the case of $[g]\varphi$, we use a dual argumentation.

- Assume that the formula has the form $\langle a \rangle \varphi$,

  $\Rightarrow$ if $[\![\mathcal{P}]\!], (p, v), \mathit{Val} \models_t \langle a \rangle \varphi$, then there is $(p, v) \xrightarrow{a} (p', v')$ such that $[\![\mathcal{P}]\!], (p, v'), \mathit{Val} \models_t \varphi$ with $v' = v[h_a := 0]$. By the induction hypothesis, $\langle\!|\mathcal{P}|\!\rangle^M, (p', v'), \mathit{Val} \models_g \varphi$. But if $(p, v) \xrightarrow{a} (p', v')$ is a transition of $[\![\mathcal{P}]\!]$ then, there is a transition $p \xrightarrow{g,a} p'$ in $\mathcal{P}$ for which $v \in [\![g]\!]$. According to the definition of $\langle\!|\mathcal{P}|\!\rangle^M$, there is also the transition $(p, v) \xrightarrow{a} (p', v')$ in $\langle\!|\mathcal{P}|\!\rangle^M$. It follows that $\langle\!|\mathcal{P}|\!\rangle^M, (p, v), \mathit{Val} \models_g \langle a \rangle \varphi$.

  $\Leftarrow$ if $\langle\!|\mathcal{P}|\!\rangle^M, (p, v), \mathit{Val} \models_g \langle a \rangle \varphi$ then there is $(p, v) \xrightarrow{a} (p', v')$ such that $[\![\mathcal{P}]\!], (p, v'), \mathit{Val} \models_g \varphi$ with $v' = v[h_a := 0]$. By the induction hypothesis, $[\![\mathcal{P}]\!], (p', v'), \mathit{Val} \models_t \varphi$. Because $(p, v) \xrightarrow{a} (p', v')$ belong to $[\![\mathcal{P}]\!]$, we get that $[\![\mathcal{P}]\!], (p, v), \mathit{Val} \models_t \langle g \rangle \varphi$.

- A dual argumentation holds in the case of $[a]\varphi$.

□

Using bisimilarity between $\langle\!\langle \mathcal{P} \rangle\!\rangle^M_{reg}$ and $\langle\!\langle \mathcal{P} \rangle\!\rangle^M$, for sufficiently big M, and that every formula is equivalent to some rectangular formula (see Proposition 127) we get the following lemma.

Lemma 133 For every process $\mathcal{P}$, for every formula φ, for every $M \geq \max(M_\varphi, M_\mathcal{P})$: $[\![\mathcal{P}]\!], (p, v), \mathit{Val} \vDash_t \varphi$ if and only if $\langle\!\langle \mathcal{P} \rangle\!\rangle^M_{reg}, (p, [v]_M), \mathit{Val} \vDash_g \mathit{Rect}_M(\varphi)$.

Theorem 134 is nothing else but a consequence of Lemma 133 and Theorem 25 as our model-checking procedure is just the one of the μ-calculus over $(\mathit{Agds}_\Sigma(M) \cup \Sigma)$-labelled transition systems.

Theorem 134 *There is an exponential time procedure that checks whether a process is a model of a formula.*

4.3 Satisfiability of the C-WT$_\mu$ Fragment

In this section we consider the satisfiability problem for C-WT$_\mu$ formulas. We will show that it is decidable whether a C-WT$_\mu$ formula has a model. We recall that formulas of the form $\langle g \rangle [a] \varphi$ or more generally, $\langle g \rangle \varphi$ where, φ is a boolean combination of formulas containing a formula of the form $[a]\psi$ (with $\psi \neq tt$), are not admitted as C-WT$_\mu$ formulas. We use tableau-based method.

4.3.1 Tableaux

We present the tableau system of rules for C-WT$_\mu$. We will assume that formulas are M-rectangular. We also define the notions of traces, μ-traces and pre-model.

Let us introduce a notation. Given a set of formulas Γ and a region r, we define the set $\Gamma_r = \{(g)\varphi \mid r \subseteq g\}$.

Definition 135 (Tableau system of rules) Let a φ be a C-WT$_\mu$ formula and let $\mathcal{D}_\varphi$ be its binding function. We define the system of tableau rules $\mathcal{S}^\varphi_c$ parametrised by φ, its binding function and the set of regions $\mathcal{R}eg_M$:

$$\frac{\{ff\}; ff}{\{\varphi, \Gamma\}; ff}(ff_r) \qquad \frac{\{ff\}; ff}{\{\langle g \rangle \varphi, \Gamma\}; r \text{ s.t } [\![g]\!] \cap r\uparrow = \emptyset}(fte) \qquad \frac{\{\Gamma\}; r}{\{[g]\varphi, \Gamma\}; r \text{ s.t } [\![g]\!] \cap r\uparrow = \emptyset}(wtt)$$

$$\frac{\{\varphi_1, \Gamma\}; r \qquad \{\varphi_2, \Gamma\}; r}{\{\varphi_1 \vee \varphi_2, \Gamma\}; r}(\vee) \qquad \frac{\{\varphi_1, \varphi_2, \Gamma\}; r}{\{\varphi_1 \wedge \varphi_2, \Gamma\}; r}(\wedge)$$

$$\frac{\{\varphi(X), \Gamma\}; r}{\{\mu X.\varphi(X), \Gamma\}; r}(\mu) \qquad \frac{\{\varphi(X), \Gamma\}; r}{\{\nu X.\varphi(X), \Gamma\}; r}(\nu) \qquad \frac{\{\varphi(X), \Gamma\}; r}{\{X, \Gamma\}; r}(reg) \quad \mathcal{D}_\varphi(X) = \sigma X.\varphi(X)$$

$$\frac{\{\varphi \mid (g)\varphi \in \Gamma_{r_i}\}; r_i \quad \forall r_i \in r\uparrow \cap \mathcal{R}eg_M}{\Gamma; r}(delay)$$

$$\frac{\varphi \cup \{\psi \mid [a]\psi \in \Gamma\}; r[h_a := 0] \text{ for each } \langle a\rangle\varphi \in \Gamma}{\Gamma; r}(mod)$$

When applying the rule (*delay*) we require that every formula in the conclusion should be in one of the forms *ff*, *tt*, $\langle g\rangle\psi$ or $[g]\psi$ and when applying the rule (*mod*) we require that every formula in the conclusion should be in one of the forms *ff*, *tt*, $\langle a\rangle\psi$ or $[a]\psi$.

Definition 136 (Tableau) A tableau for a formula φ from a region r^0 is a pair $\tau^{\varphi}_{r^0} = \langle \mathcal{T}, \mathcal{L}\rangle$, where $\mathcal{T} = \langle N, E\rangle$ is a tree, and $\mathcal{L}$ is a labeling function such that:

1. The root n^0 of $\tau^{\varphi}_{r^0}$ is labeled by $\{\varphi\}; r^0$
2. The sons of any node n are created and labeled according to the rules of systems $\mathcal{S}^{\varphi}$. It is required the rules (*mod*) and (*delay*) are applied only when no other rule is applicable.

Given a node n such that $\mathcal{L}(n) = \Gamma; r$, $\mathcal{L}_{\mathcal{F}}(n) = \Gamma$ and $\mathcal{L}_{\rho}(n) = r$ denote the *formula part* and the *timing part* of $\mathcal{L}(n)$.

If we construct a tableau for a C-WT$_\mu$ formula, a conclusion never contains at the same time a formula starting with a modality indexed with a guard and a formula starting with a modality indexed with an event. So, in a tableau, the formula part of timed sequents on which no rule is applicable never contain formulas of the forms $\langle a\rangle\psi$, $\langle g\rangle\psi$ and $[g]\psi$.

Definition 137 (Trace) Given a path π of $\tau^{\varphi}_{r^0} = \langle \mathcal{T}, \mathcal{L}\rangle$, a *trace* on π is a function $\mathtt{F}$ that assigns a tuple made of a formula and a region to each node in some initial segment of π, according to the rules applied for the construction of π. $\mathtt{F}_{\mathcal{F}}$ and $\mathtt{F}_{\rho}$ denotes the *formula part* and the *timing part* of $\mathtt{F}(n)$. $\mathtt{F}$ satisfies the following conditions:

1. if $\mathtt{F}(n)$ is defined then $\mathtt{F}_{\mathcal{F}}(n) \in \mathcal{L}_{\mathcal{F}}(n)$ and $\mathtt{F}_{\rho}(n) = \mathcal{L}_{\rho}(n)$;
2. if the rule applied at the node m is not directed by $\mathtt{F}(m)$ then the son $n \in \pi$ of m is such that $\mathtt{F}(m) = \mathtt{F}(n)$;
3. if the rule is directed by $\mathtt{F}(m)$ but it is not (*mod*), then the tuple $\mathtt{F}(n)$ for the son $n \in \pi$ of m is one of the results of the application application of the rule;
4. if the rule (*delay*) is applied at the node m and the son $n \in \pi$ of m is labeled by $\{\varphi \mid (g)\varphi \in \Gamma_{r_i}\}; r_i$ then:
   - $\mathtt{F}(n)$ is equal to $\varphi; r_i$ if $\mathtt{F}(m) = (g)\varphi; r$;
   - otherwise $\mathtt{F}(n)$ is undefined;
5. if the rule (*mod*) is applied at m and the son $n \in \pi$ of m is labeled by $\varphi \cup \{\psi \mid [a]\varphi \in \Gamma\}; r[h_a := 0]$ for some $\langle a\rangle\varphi \in \Gamma$ then:
   - $\mathtt{F}(n) = \varphi; r[h_a := 0]$ if $\mathtt{F}(m) = \langle a\rangle\varphi; r$;
   - $\mathtt{F}(n) = \psi; r[h_a := 0]$ if $\mathtt{F}(m) = [a]\psi; r$;
   - otherwise $\mathtt{F}(n)$ is undefined and $\mathtt{F}(m)$ is the last element of the trace.

A variable X is regenerated on a trace $\mathtt{F}$ of some path if and only if for some m and its son n on the path $\mathtt{F}_{\mathcal{F}}(m) = X$ and $\mathtt{F}_{\mathcal{F}}(n) = \psi(X)$ with $\mathcal{D}_\varphi(X) = \sigma X.\psi(X)$.

A *μ-trace* is a infinite trace on which the oldest variable regenerated infinitely often is a *μ-variable*; or a maximal finite trace, ending with a tuple the formula part of which contains *ff*.

An *pre-model* $\mathcal{PM}$ is a fragment of a tableau $\tau^{\varphi}_{r_0}$ satisfying the following conditions:

- the root of $\tau^{\varphi}_{r_0}$ belongs to $\mathcal{PM}$;
- if a disjunctive node belongs to $\mathcal{PM}$, then only one of its sons belongs to $\mathcal{PM}$;
- for all other kinds of nodes, if a node belongs to $\mathcal{PM}$ then all its successors too;
- there is no path with a μ-trace in $\mathcal{PM}$.

The notions of signature, μ-signature, and ν-signature are defined as in Chapter 3, Definition 104. The proof of the following lemma is the same as the analogous lemma (Lemma 105) in Chapter 3.

Lemma 138 (μ-Signature) Let ${}^{\mu}\mathtt{sig}(\varphi, s)$ the signature of φ at s, it is true that:

- ${}^{\mu}\mathtt{sig}(\varphi_1 \wedge \varphi_2, s) = \max\{{}^{\mu}\mathtt{sig}(\varphi_1, s), {}^{\mu}\mathtt{sig}(\varphi_2, s)\}$
- ${}^{\mu}\mathtt{sig}(\varphi_1 \vee \varphi_2, s) = {}^{\mu}\mathtt{sig}(\varphi_1, s)$ or ${}^{\mu}\mathtt{sig}(\varphi_1 \vee \varphi_2, s) = {}^{\mu}\mathtt{sig}(\varphi_2, s)$
- ${}^{\mu}\mathtt{sig}(\langle a\rangle\varphi, s) = {}^{\mu}\mathtt{sig}(\varphi, s')$ for some s' such that $s \xrightarrow{a} s'$
- ${}^{\mu}\mathtt{sig}([a]\varphi, s) = \max\{{}^{\mu}\mathtt{sig}(\varphi, s')$ for all s' such that $s \xrightarrow{a} s'\}$
- ${}^{\mu}\mathtt{sig}(\langle g\rangle\varphi, s) = {}^{\mu}\mathtt{sig}(\varphi, s')$ some s' such that $s \xrightarrow{g} s'$
- ${}^{\mu}\mathtt{sig}([g]\varphi, s) = \max\{{}^{\mu}\mathtt{sig}(\varphi, s')$ for all s' such that $s \xrightarrow{g} s'\}$
- If X_i is the $i-th$ variable of $\mathcal{D}_\varphi$ and $\mathcal{D}_\varphi(X_i) = \mu X_i\varphi(X_i)$, then the prefix of length $i-1$ of ${}^{\mu}\mathtt{sig}(\mu X_i.\varphi(X_i), s)$ and ${}^{\mu}\mathtt{sig}(\varphi(X), s)$ are equal
- ${}^{\mu}\mathtt{sig}(\nu X.\varphi(X), s) = {}^{\mu}\mathtt{sig}(\varphi(X), s)$ where $\mathcal{D}_\varphi(X) = \nu X.\varphi(X)$
- If $\mathcal{D}_\varphi(Y) = \mu Y.\varphi(Y)$, then ${}^{\mu}\mathtt{sig}(Y, s) > {}^{\mu}\mathtt{sig}(\varphi(Y), s)$
- If $\mathcal{D}_\varphi(Y) = \nu Y.\varphi(Y)$, then ${}^{\mu}\mathtt{sig}(Y, s) = {}^{\mu}\mathtt{sig}(\varphi(Y), s)$

Properties for the ν-signature ${}^{\nu}\mathtt{sig}(\varphi, s)$ can be defined in a dual way. An analogous lemma to Lemma 138 considers $(\mathcal{V}_\Sigma \cup \Sigma)$-LTS and $\vDash_t$ instead of $(Gds_\Sigma \cup \Sigma)$-LTS and $\vDash_g$.

4.3.2 Satisfiability Results

We show in Theorem 139 that a formula is satisfiable if and only if its tableau contains a pre-model. In Proposition 142 we show that a formula has a model if there is a pre-model in its tableau and in Proposition 140 we show that if a formula has a model then there is a pre-model in its tableau. The complexity of the satisfiability checking is the same as the complexity of checking the existence of a pre-model in a tableau.

Theorem 139 *There is an exponential time procedure in the size of the formula that checks if a formula φ is satisfiable.*

The proof is a consequence of the two following propositions.

Proposition 140 If φ is satisfiable then there is a pre-model in $\tau^{\varphi}_{r^0}$.

Proof
If φ is satisfiable, then there exists a process $\mathcal{P}$ such that $\mathcal{P} \vDash_t \varphi$. Without the loss of generality we can assume that φ is rectangular. By Lemma 133, for every $M \geq \max(M_{\mathcal{P}}, M_{\varphi})$, we have that $(\!|\mathcal{P}|\!)^M_{reg} \vDash_g \varphi$.

Consider $\tau^{\varphi}_{r^0}$ the tableau for φ; then we choose the nodes of $\tau^{\varphi}_{r^0}$ that we include in the pre-model $\mathcal{PM}$ accordingly to a *marking relation* $M : N \to 2^S$. It will be defined in such a way that $s \in M(n)$ implies $(\!|\mathcal{P}|\!)^M_{reg}, s, Val \vDash_g (\!|\psi|\!)_{\mathcal{D}_\varphi}$ for every $\varphi \in \mathcal{L}_{\mathcal{F}}(n)$. First, we put s^0 in $M(n^0)$ with n^0 being the root of τ_φ. This is consistent as $(\!|\mathcal{P}|\!)^M_{reg} \vDash_g \varphi$.

Then, if we assume that the node n has been included in the pre-model $\mathcal{PM}$ with $s_n \in M(n)$, we choose the next node to include in the tableau using the following rules:

- The only son n' of some node n, marked with s_n, on which an unary rule ($wtt, \wedge, reg, \mu$, or ν) was applied is included in $\mathcal{PM}$ and we set $s_n \in M(n')$.
- If n is a disjunctive node, then s_n is put into the marking of the son for which it has the least μ-signature. By Lemma 138, such a son exists.
- If n is a delay node, then we add all the sons of n in $\mathcal{PM}$. Each son n' of n is the result of the reduction of a set of formulas of the form $\langle g\rangle\psi$ or $[g]\psi$ with respect to a region r_i. Then, we set $s_{n'} \in M(n')$ where $s_{n'}$ is the unique state such that $s_n \xrightarrow{g} s_{n'}$.
- If n is a modal node, then we add all the sons of n in $\mathcal{PM}$. Each son n' of n is the result of the reduction of a formula of the form $\langle a\rangle\psi$. Then, we set $s_{n'} \in M(n')$ where $s_{n'}$ is a state such that $s_n \xrightarrow{a} s_{n'}$ and ${}^{\mu}\mathtt{sig}(\langle a\rangle\varphi, s_n) \geq^{\mu} \mathtt{sig}(\varphi, s_{n'})$. By Lemma 138, such a son exists.

Using similar argumentation described at the end of the proof of Proposition 108 we can show that every path in the above pre-model does not contain a μ-trace. □

From the definition of the tableau rules, applying a rule different from (*mod*), (*delay*) and ($\vee$) to a node of a tableau generates a unique successor. In a symbolic pre-model we choose only one son of a disjunctive node and all the sons of a modal or a delay node. It follows that in a symbolic pre-model, the nodes with more that one successor are modal or delay nodes.

Given a node n of $\mathcal{PM}$ we denote $des^\alpha(n)$ the closest descendant of n or n itself in $\mathcal{PM}$ that is either a delay node, a modal node, or a leaf. Observe that, if n is the root of $\mathcal{PM}$ or n is a successor of a modal node of $\mathcal{PM}$, then $des^\alpha(n)$ is a delay node or a leaf; if n is a successor of a delay node of $\mathcal{PM}$, then $des^\alpha(n)$ is a modal node.

Definition 141 (model for a pre-model) Given a pre-model $\mathcal{PM}$ for a formula φ, the *model based on* $\mathcal{PM}$ is the timed process $\mathcal{P} = \langle P, \Sigma, p^0, \Delta_P \rangle$ such that:

1. $p^0 = des^\alpha(n^0)$ where n^0 is the root of $\mathcal{PM}$.
2. P consists of all the leaves and delay nodes of $\mathcal{PM}$.
3. $(p, g, a, des^\alpha(n')) \in \Delta_P$ if there is in $\mathcal{PM}$ a successor n of p obtained by reducing a region $r_i \subseteq g$ with $g \in Agds(M_\varphi)$ and a successor n' of $des^\alpha(n)$ obtained by reducing an action a.

Remark: From the definition above, the maximal constant that occurs in the model $\mathcal{P}$ constructed from a pre-model $\mathcal{PM}$ is the same as maximal constant that occurs in the formula. Moreover, the constraints in the model are rectangular.

Proposition 142 Given a formula φ, if there is a pre-model in $\tau^\varphi_{r^0}$, then φ is satisfiable.

Proof
Assume that φ has a pre-model $\mathcal{PM}$ and φ is not satisfiable. Let $M = M_\varphi$. Consider $\mathcal{P}$, the model based on $\mathcal{PM}$. From the remark above, $M_\mathcal{P} = M_\varphi$. If $\mathcal{P} \not\Vdash \varphi$, then by Lemma 133 we get that, $\langle\!|\mathcal{P}|\!\rangle^M_{reg} \not\Vdash_g \varphi$. If so, we show that $\mathcal{PM}$ contains a path π with a μ-trace $\mathtt{F} = \{\varphi_m; r_m\}_{m \in \pi}$. The path π and the trace $\mathtt{F}$ are built in the following way:

- π starts at m_0 and $\varphi_{m_0} = \varphi$.
- Assume that, we built $\mathtt{F}$ up to the tuple $\varphi_m; r_m$ with, $\varphi_m \in \mathcal{L}_\mathcal{F}(m)$ and $r_m = \mathcal{L}_\rho(m)$, such that $\langle\!|\mathcal{P}|\!\rangle^M_{reg}, (des^\alpha(m), r_m) \not\Vdash_g \langle\!|\varphi_m|\!\rangle_{\mathcal{D}_\varphi}$. The formula of the next tuple (the timing part is obvious) is selected as follows:
  1. If m is not a delay nor a modal node, then the only son m' of m is such that
     - $\mathcal{L}_\rho(m) = \mathcal{L}_\rho(m')$ and there are equal to r_m.
     - if φ_m was not reduced by the rule then $\varphi_{m'} = \varphi_m$.
     - if $\varphi_m = \varphi_1 \wedge \varphi_2$ is reduced then $\varphi_{m'} = \varphi_1$ if $^\nu\mathtt{sig}(\varphi_m, (des^\alpha(m), r)) \geq^\nu \mathtt{sig}(\varphi_1, (des^\alpha(m), r))$, otherwise $\varphi_{m'} = \varphi_2$.
     - if $\varphi_m = \varphi_1 \vee \varphi_2$ is reduced then $\varphi_{m'}$ is the formula that occurs in $\mathcal{L}_\mathcal{F}(m')$. We remark that, the choice in this case is directed by $\mathcal{PM}$.
     - In the other sub cases (i.e $ff, fte, wtt, \mu, \nu,$, or reg), we just take the resulting formula as the one for the next tuple of the trace.
  2. If m is a delay node and φ_m is of the form $\langle g \rangle \psi$ or $[g]\psi$ and there is a son m' of m the formula part of which contains ψ, then we take $\varphi_{m'} = \psi$.
  3. If m is a modal node, it is necessarily the closest descendant of a successor n' (with respect to some region r_m) of some delay node n; then,

- if $\varphi_m = \langle a \rangle \psi$, there is a son m' of m the formula part of which was obtained by reducing φ_m, and the timing part of which is $r_m[h_a := 0]$. We take $\varphi_{m'} = \psi$.
- if $\varphi_m = [a]\psi$, then because $(des^\alpha(m), r_m) \not\models_g \langle\!\langle \varphi_m \rangle\!\rangle_{\mathcal{D}_\varphi}$, there exists a state p', a constraint g such that $des^\alpha(n) \xrightarrow{g,a} p'$ is a transition in $\mathcal{P}$ and ${}^\nu\mathtt{sig}([a]\psi, (des^\alpha(n), r_m)) =^\nu \mathtt{sig}(\psi, (p', r_m[h_a := 0])$ with $r_m \subseteq g$ and $g \in Agds(M_\varphi)$. We take $\varphi_{m'} = \psi$ and $r_{m'} = r_m[h_a := 0]$.

We remark that $\mathtt{F}$ is a valid trace of $\mathcal{PM}$ and we distinguish two cases:

1. The trace is finite;
   - If the trace ends with the formula *ff*, then we get a contradiction with that $\mathcal{P}$ derived from $\mathcal{PM}$; Indeed a trace of $\mathcal{PM}$ never ends with *ff*.
   - If the trace ends at the node m with the formula $\varphi_m = tt$, then m is a leaf or a delay node and obviously, $\langle\!\langle \mathcal{P} \rangle\!\rangle^M_{reg}, (des^\alpha(m), r_m) \models_g tt$, leading to a contradiction with the hypothesis.
   - If the trace ends with a formula of form $[g]\varphi$, then the region at node m could never reach g meaning that $[g]\varphi$ is satisfied at m. We also get a contradiction with our hypothesis.
   - Assume that the trace ends at the node m with a formula of the form $[a]\varphi_c$. There is the closest ancestor n of node m which is a delay node. The selected formula at the node n that occurs in the trace has the form $[g]\psi$ or $\langle g \rangle \psi$ and ψ is a boolean combination of formulas containing $[a]\varphi_c$.
     Let p be the state in $\mathcal{P}$ that corresponds to the node n. Such a state exits because n is a delay node. Let r be the region at the node m and r' be region at the node n. Because m is a son of n, we have that $r \in r'\uparrow$. Additionally, by hypothesis, $\langle\!\langle \mathcal{P} \rangle\!\rangle^M_{reg}, (p, r') \not\models_g \langle\!\langle (g)\psi \rangle\!\rangle_{\mathcal{D}_\varphi}$. Because the trace is maximal, there is no transition from p labelled with (g, a) for the unique constraint $g \in Agds(M)$ such that $r \subseteq g$. It follows that in $\langle\!\langle \mathcal{P} \rangle\!\rangle^M_{reg}$ there a unique outgoing transition $(p, r') \xrightarrow{g} (p, r)$ and there is no outgoing transition from (p, r) labelled with a. This implies $\langle\!\langle \mathcal{P} \rangle\!\rangle^M_{reg}, (p, r) \models_g \langle\!\langle [a]\varphi_c \rangle\!\rangle_{\mathcal{D}_\varphi}$. Contradiction with that in the trace ${}^\nu\mathtt{sig}((g)\psi, (p, r')) = {}^\nu\mathtt{sig}([a]\varphi, (p, r))$. Indeed, recall that the trace has been built by choosing at every node, the formula and the configuration with the least ν-signature.
2. If the trace is infinite, then because the ν-signature decreases along the trace and the formula is of finite length, there is necessarily a μ-variable X that is infinitely often regenerated and no older variable than X is infinitely often regenerated. This is a contradiction with that $\mathcal{PM}$ does not contain a μ-trace.

□

Proposition 140 and Proposition 142 ends the proof of Theorem 139.

4.3.3 Existence of Deterministic Models for Formulas

We may wonder if a formula has a deterministic model. The solution to that question is difficult as deterministic models may need constants that are strictly greater than the maximal constant

occurring in formulas. This is the case with the following formula. Let φ be the formula defined by:

$$\varphi = (\langle h_a > 1\rangle\langle a\rangle\langle 0 < h_a < 1\rangle\langle c\rangle tt) \wedge (\langle h_a > 1\rangle\langle a\rangle[0 < h_a < 1][c]ff).$$

Observe that φ can be rewritten using the syntax of ERL; so, the same problem appears for ERL. The formula φ says that there are two time instants satisfying $h_a > 1$ at which event a must occur. In one of these time instants the event c must occur when $0 < h_a < 1$ is satisfied and for the other time instant the event c never occurs in time instants satisfying $0 < h_a < 1$. Using our procedure for the satisfiability, the resulting model will be of the form of the timed process in Figure 16 with g_1 and g_2 are instantiated to $h_a > 1$. Model in Figure 16 is deterministic if conjunction of g_1 and g_2 is inconsistent. For example, g_1 and g_2 could be respectively instantiated to $1 < h_a < 3$ and $4 < h_a$.

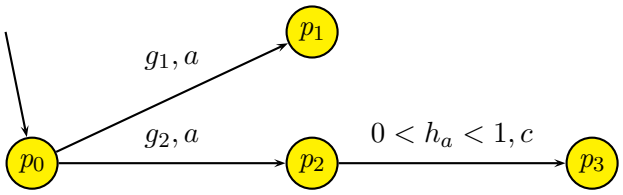

Figure 16: A Timed process.

We can also consider a situation when we impose a maximal constant with which the clocks can be compared in the models. Such a constant can be greater than the maximal constant occurring in the formula. Under such an assumption, checking the existence of a deterministic models for WG-WT$_\mu$ formulas (not only C-WT$_\mu$ formulas) is done by replacing the rule (*mod*) of the tableau by the following rule:

$$\frac{\{\psi \,|\, (a)\psi \in \Gamma\}; r[h_a := 0] \text{ for each } a \text{ s.t } \langle a\rangle\varphi \in \Gamma}{\Gamma; r}(moddet)$$

We remark that our tableau system of rules for the M-bounded satisfiability checking problem of WG-WT$_\mu$ is the same as the tableau system of rules for the satisfiability of C-WT$_\mu$; only the nature of the formulas in the timed sequents changes.

If we consider the formula φ presented just above and we execute our satisfiability procedure for checking whether φ has a deterministic model of bound $M = 1$, it will result that φ does not have a deterministic M-bounded model. This is because in our models clocks are compared with integer. But, if we check for models of bound $M = 2$, our satisfiability decision procedure will return that φ has a deterministic model.

4.4 Concluding Remarks

We have defined a new logic called WT$_\mu$ that is interpreted over timed processes. The logic WT$_\mu$ is a "weak" real-time extension of the μ-calculus. The logic WT$_\mu$ is expressive enough to handle the necessity modal operator on time delay and can be used to describe properties like *"An event can be completed at every time a condition on the time is satisfied"*. We have used the region abstraction to show that it is decidable whether a timed process satisfies a formula. Our technique leads to an exponential algorithm that requires the construction of M-region

representations of timed processes. Also based on the region representation, we have shown that it is decidable whether there exists a timed process that satisfies a formula of C-WT_μ. We have shown that checking the existence of deterministic model for a formula may require the use of constants greater than the ones used in the formula. This problem can be avoided if we assume that there is a maximal value with which the clocks can be compared. These results are fundamental for the controller synthesis problems that we consider in the next chapter.

Chapter 5

Centralised Controller Synthesis using C-WT$_\mu$ Specification

It could happen that behaviours of a real-time system (called a *plant*) do not satisfy a property, because for example, either that property is the result of a modification of an initial property or, there are bugs in the real-time system; then we would like to provide a new system that satisfies the property. To tackle this problem two approaches can be considered: either we destroy the old system and we design (when it is possible) a new one, or we design (when it is possible) another system (called a *controller*) that we combine with the plant in such a way that the resulting system (called the *controlled or supervised system*) satisfies the expected property. The first approach is expensive for big systems or for systems that only need small modifications that can be done by another system. The second approach is cheap if the plant can be controlled by disabling some events and the controller is small compared to the controlled system. In this chapter we consider the second approach. Given a plant and a property, the *c*ontroller synthesis problem can be understood as the search for a component, called the controller such that the the controlled system satisfies the property. Systems are reactive and evolve in some environment. We make some convenient and practical assumptions on events. We distinguish uncontrollable events (for example, the ones that come from the environment) from controllable events (for example the ones that come from the system). We assume that, controllers can never prevent uncontrollable events to happen whatever is the time they occur.

In our framework, plants and controllers are modelled with timed processes. Properties are described with C-WT$_\mu$ formulas. The combination between a plant and a controller is the product of their models (timed processes). We define modal automata which are, roughly speaking, another way for presenting WT$_\mu$ formulas. Modal automata are interpreted over timed processes. We show that modal automata are closed under intersection. We translate WT$_\mu$ formulas into equivalent modal automata and reciprocally we translate modal automata into an equivalent WT$_\mu$ formulas. As formulas use fixpoint operators that are difficult to handle; then we use modal automata to describe properties of systems. We consider subclasses of modal automata that we called *well guarded modal automata* (WG-MA) and modal automata for control (C-MA). The class of WG-MA is equivalent to class of WG-WT$_\mu$ formulas, and the class C-MA is equivalent to the class of C-WT$_\mu$ formulas. In consequence, the emptiness checking problem for C-MA is the same as the satisfiability checking problem of C-WT$_\mu$

formulas. Both problems are decidable.

We consider two controller synthesis problems: the centralised controller synthesis problem and the Δ-dense-time control.

The centralised controller synthesis problem (CCP) is the following:

(CCP) *Given the model of a plant $\mathcal{P}$ and a property described with a C-MA $\mathcal{A}$, does there exits a controller $\mathcal{R}$ such that $\mathcal{P} \times \mathcal{R} \models \mathcal{A}$ and satisfying also the following control condition (CC)*:

Control condition(CC) $\mathcal{R}$ does not restrict environment events.

Our solution to CCP consist to define the *quotient* of automata by the plants. The result of that operation is a modal automaton that the controller must satisfy in addition to the control condition. The control condition will be described with a modal automaton. Then, a controller will be the model resulting from the satisfiability checking procedure of an automaton which is semantically equivalent to the intersection of the quotient automaton with an automaton that describes the control condition.

The Δ-dense-time control amounts to finding a controller (also called a Δ-controller) for a system such that at least $\Delta \geq 0$ time units elapse between two consecutive controllable events. We will show that this problem is decidable and it is a corollary to (CCP) as properties on a Δ-controller can be described in a C-MA. We will be able to construct a witness Δ-controller.

Related results: The controller synthesis problems have been introduced by Ramadge and Wonham [RW89] and since then, they have been considered by many authors in the settings of untimed systems [PR05, BK06, AW07] and timed systems [Sav01, DM02, BDMP03, BCL05]. The framework of Arnold et al. [AVW03, AW07] is a considerable extension of the framework of Ramadge et al. as it considers branching-time properties and the μ-calculus is expressible enough for describing the control condition. Arnold et al. [AVW03, AW07] also use modal automata instead of formulas for describing specifications. These modal automata are some kind of alternating automata [Tho97] over labelled trees. The Madhusudan et al. [DM02, BDMP03] framework for the controller synthesis is an extension of the framework of Ramadge and Wonham [RW89] and Pnueli et al. [AMP95] to real-time systems modeled with timed automata and it does not consider more general timed branching-time properties. In the framework of Laroussinie et al. [BCL05] the logic L_ν is used for the centralised controller synthesis and the Δ-dense-time control of timed automata. The solution provided in [BCL05] gives an answer to the existence of a controller; it does not show how to build controllers. This is because the satisfiability problem of L_ν are still open. Our contributions place themselves between the framework of Arnold et al. and the framework of Laroussinie et al. as our model is a subclass of timed automata and we use the techniques of Arnold et al.

The chapter is organized as follows: In the next section, we define modal automata, their semantics; we show that they are closed under intersection and we present an algorithm for the model-checking of modal automata. The translation between modal automata and WT$_\mu$ formulas is presented in Section 5.2. In Section 5.3, we define well guarded modal automata (WG-MA) and modal automata for control (C-MA). We define the quotient of WG-MA by timed processes that we use in Section 5.4 for controller synthesis.

5.1 Modal Automata and Modal Automata for Controller Synthesis

We define modal automata that are interpreted over timed processes. The interpretation of a modal automaton is presented as a two player parity game and acceptance is defined in terms of winning strategy for a player in that game. We show that modal automata are closed under intersection and we consider the model-checking problem for modal automata. This problem is reduced to checking the existence of a winning strategy in the acceptance game. We will also define subclasses of modal automata. The first one is the subclass of well guarded modal automata (WG-MA) and the second subclass is the one modal automata for control (C-MA).

5.1.1 Definition and Semantics

Modal automata are nothing else but WT_μ formulas written in the automata syntax; they are kinds of timed alternating automata with parity acceptance condition. They use modal formulas in their transition relations.

Definition 143 The set of *modal formulas* over Σ and Q, denoted $\mathcal{MF}(\Sigma, Q)$ is the smallest set closed under the following rules:

- tt, ff, q are modal formulas, where $q \in Q$.
- $\varphi \vee \psi$, $\varphi \wedge \psi$ are modal formulas for all $\varphi, \psi \in \mathcal{MF}(\Sigma, Q)$.
- $\langle g \rangle \psi$, $\langle a \rangle \psi$, $[g]\psi$, $[a]\psi$ are modal formulas for all $\psi \in \mathcal{MF}(\Sigma, Q)$ where $g \in Gds_\Sigma$.

Definition 144 A *modal automaton* (MA for short) over Σ is a tuple

$$\mathcal{A} = \langle Q, \Sigma, q^0, \Delta_\mathcal{A} : Q \to \mathcal{MF}(\Sigma, Q), Acc_\mathcal{A} \subseteq Q^\omega \rangle$$

where:

- Q is a finite set of states.
- $q^0 \in Q$ is the initial state.
- $\Delta_\mathcal{A}$ is a transition relation.
- $Acc_\mathcal{A}$ is the max-parity condition given by the parity function $rank : Q \to \mathbb{N}$.

Modal automata accept timed processes and their semantics is defined using acceptance games.
We define the *real-time acceptance game* of a timed process $\mathcal{P} = \langle P, \Sigma, p^0, \Delta_P \rangle$ and a modal automaton $\mathcal{A}$. Let $\mathcal{F}$ be the set of formulas containing all the formulas appearing as values of transition function $\Delta_\mathcal{A}$ and closed under sub formulas. The game $\mathcal{G}(\mathcal{P}, \mathcal{A})$ is $\langle N_E, N_A, T, Acc_\mathcal{G} \rangle$ where:

- $N_E = P \times \mathcal{F}_E^\mathcal{A} \times \mathcal{V}$ and $\mathcal{F}_E^\mathcal{A} \subseteq \mathcal{F}$ is a the set of modal formulas of the form ff, $\varphi \vee \psi$, $\langle g \rangle \varphi$, $\langle a \rangle \varphi$.

- $N_A = (P \times \mathcal{F} \times \mathcal{V}) \setminus N_E$.
- There is no move from either (p, tt, v) or (p, ff, v) for every $v \in \mathcal{V}$.
- From $(p, \varphi \wedge \psi, v)$ as well as from $(p, \varphi \vee \psi, v)$ there are moves to (p, φ, v) and to (p, ψ, v).
- From $(p, [g]\varphi, v)$ and from $(p, \langle g \rangle \varphi, v)$ there are moves to $(p, \varphi, v + t)$ for every t such that $v + t \in [\![g]\!]$.
- From $(p, [a]\varphi, v)$ and from $(p, \langle a \rangle \varphi, v)$ there are moves to $(p', \varphi, v[h_a := 0])$ for every p' such that $p \xrightarrow{g,a} p'$ and $v \in [\![g]\!]$.
- There is a move from (p, q, v) to $(p, \Delta(q), v)$.
- $Acc_{\mathcal{G}}$ is the set of infinite sequences projection of which on Q is in Acc.

We say that $\mathcal{A}$ accepts $\mathcal{P}$ (or $\mathcal{P}$ is a model of $\mathcal{A}$) and we write $\mathcal{P} \in \mathcal{L}(\mathcal{A})$ or $\mathcal{P} \models \mathcal{A}$ if and only if there is a winning strategy for the player *Eve* from the position (p^0, q^0, v^0) in $\mathcal{G}(\mathcal{P}, \mathcal{A})$. We define the language of an automaton $\mathcal{A}$ denoted by $\mathcal{L}(\mathcal{A})$, as the set of processes it accepts. Formally

$$\mathcal{L}(\mathcal{A}) = \{\mathcal{P} \mid \mathcal{P} \models \mathcal{A}\}$$

We show that modal automata are closed under intersection. Let us define an intersection operation between modal automata.

Definition 145 Let

$$\mathcal{A}_1 = \langle Q_1, \Sigma, q_1^0, \Delta_1 : Q_1 \to \mathcal{F}(\Sigma, Q_1), Acc_1 \subseteq Q_1^\omega \rangle$$

and

$$\mathcal{A}_2 = \langle Q_2, \Sigma, q_2^0, \Delta_2 : Q_2 \to \mathcal{F}(\Sigma, Q_1), Acc_2 \subseteq Q_2^\omega \rangle$$

be two modal automata. Consider the automaton

$$\mathcal{A}_1 \wedge \mathcal{A}_2 = \langle Q, \Sigma, q^0, \Delta : Q \to \mathcal{F}(\Sigma, Q), Acc \subseteq Q^\omega \rangle$$

where

- $Q = Q_1 \cup Q_2 \cup \{q^0\}$
- q^0 is the initial state.
- Δ is defined by:

$$\Delta(q) = \begin{cases} \Delta_1(q) & \text{if } q \in Q_1 \\ \Delta_2(q) & \text{if } q \in Q_2 \\ \Delta_1(q_1^0) \wedge \Delta_2(q_2^0) & \text{if } q = q^0 \end{cases}$$

- $Acc = Acc_1 \cup Acc_2$

Proposition 146 Given two automata $\mathcal{A}_1$ and $\mathcal{A}_2$, for every timed process $\mathcal{P}$, $\mathcal{P} \in \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ if and only if $\mathcal{P} \in \mathcal{L}(\mathcal{A}_1 \wedge \mathcal{A}_2)$.

Proof
If $\mathcal{P} \in \mathcal{L}(\mathcal{A}_1 \wedge \mathcal{A}_2)$, then Eve has a winning strategy in $\mathcal{G}(\mathcal{P}, \mathcal{A}_1 \wedge \mathcal{A}_2)$, meaning that Eve wins every play starting from (p^0, q^0, v^0) and consistent with that strategy. From (p^0, q^0, v^0) there is a move to $n_0 = (p^0, \Delta_1(q_1^0) \wedge \Delta_2(q_2^0), v^0)$. There are moves from n_0 to $n_1 = (p^0, \Delta_1(q_1^0), v^0)$ and $n_2 = (p^0, \Delta_2(q_2^0), v^0)$ and Eve has a winning strategy from n_1 and a winning strategy from n_2. These strategies are also winning for Eve in $\mathcal{G}(\mathcal{P}, \mathcal{A}_1)$ and $\mathcal{G}(\mathcal{P}, \mathcal{A}_2)$.
Conversely, from winning strategies in $\mathcal{G}(\mathcal{P}, \mathcal{A}_1)$ and $\mathcal{G}(\mathcal{P}, \mathcal{A}_2)$, the winning strategy in $\mathcal{G}(\mathcal{P}, \mathcal{A}_1 \wedge \mathcal{A}_2)$ mimics either the winning strategy in $\mathcal{G}(\mathcal{P}, \mathcal{A}_1)$ or the winning strategy in $\mathcal{G}(\mathcal{P}, \mathcal{A}_2)$, depending on the fist move of Eve. □

5.1.2 Model- Checking

We address the model-checking problem of modal automata which is to check if an automaton accepts a given timed process. For this question we use similar technique to the model-checking problem of WT_μ formulas in Section 4.2. As real-time acceptance game arena is infinite, we need to abstract that game in such a way that the arena of the resulting game is finite.

Symbolic acceptance game of a timed process Let $\mathcal{P} = \langle P, \Sigma, p^0, \Delta_P \rangle$ be a timed process and $\mathcal{A}$ a modal automaton as above. Let, as before, $\mathcal{F}$ be the set of formulas containing all the formulas that are the values of $\Delta_\mathcal{A}$ and all their sub formulas. The *M-symbolic acceptance game* of $\mathcal{P}$ and $\mathcal{A}$ is the structure $\hat{\mathcal{G}}(\mathcal{P}, \mathcal{A}, M) = \langle N_E, N_A, T, Acc \rangle$ where

- $N_E = P \times \mathcal{F}_E^\mathcal{A} \times Reg(M)$ and $\mathcal{F}_E^\mathcal{A} \subseteq \mathcal{F}$ is the set of modal formulas of the form ff, $\varphi \vee \psi$, $\langle g \rangle \varphi$, $\langle a \rangle \varphi$.
- $N_A = P \times \mathcal{F} \setminus N_E$.
- There is no move from (p, tt, r), nor (p, ff, r) for every $r \in \mathcal{R}eg(M)$.
- From $(p, \varphi \wedge \psi, r)$ as well as from $(p, \varphi \vee \psi, r)$ there are moves to (p, φ, r) and to (p, ψ, r).
- From $(p, [g]\varphi, r)$ and from $(p, \langle g \rangle \varphi, r)$ there are moves to (p, φ, r') for every $r' \in r\uparrow$ such that $r' \subseteq g$.
- From $(p, [a]\varphi, r)$ and from $(p, \langle a \rangle \varphi, r)$ there are moves to $(p', \varphi, r[h_a := 0])$ for every p' such that $p \xrightarrow{g,a} p'$ and $r \subseteq g$.
- There is a move from (p, q, r) to $(p, \Delta(q), r)$.
- $Acc_{\hat{\mathcal{G}}}$ is the set of infinite sequences projection of which on Q is in Acc.

We say that $\mathcal{P}$ is an M-symbolic model of $\mathcal{A}$ and we write $\mathcal{P} \vDash_M \mathcal{A}$ if and only if there is a winning strategy for the player *Eve* in $\hat{\mathcal{G}}(\mathcal{P}, \mathcal{A}, M)$.

Proposition 147 For every automaton $\mathcal{A}$, for every timed process $\mathcal{P}$, for every $M \geq \max(M_\mathcal{A}, M_\mathcal{P})$, *Eve* has a winning strategy in $\mathcal{G}(\mathcal{P}, \mathcal{A})$ if and only if *Eve* has a winning strategy in $\hat{\mathcal{G}}(\mathcal{P}, \mathcal{A}, M)$.

Proof
The choice of $M \geq \max(M_{\mathcal{A}}, M_{\mathcal{P}})$ follows from a similar argument as in the case of Lemma 133 In the setting of modal automata, acceptance is defined in terms of a parity game. We show that if *Eve* player has a winning strategy in $\mathcal{G}(\mathcal{P}, \mathcal{A})$ then she also has a winning strategy in $\hat{\mathcal{G}}(\mathcal{P}, \mathcal{A}, M)$ and reciprocally. For this, we show how a move of a player from a position (p, φ, v) to some position (p', φ', v') can be mimicked by moves of the same player from $(p, \varphi, [v]_M)$ to $(p', \varphi', [v']_M)$ and reciprocally.

We note that a play in $\mathcal{G}(\mathcal{P}, \mathcal{A})$ starts in (p^0, q^0, v^0) and a play in $\hat{\mathcal{G}}(\mathcal{P}, \mathcal{A}, M)$ starts in (p^0, q^0, r^0) with $r^0 = [v^0]_M$.

Assume that the current position in $\mathcal{G}(\mathcal{P}, \mathcal{A})$ is $n = (p, \varphi, v)$, and the current position in $\hat{\mathcal{G}}(\mathcal{P}, \mathcal{A}, M)$ is $\overline{n} = (p, \varphi, r)$ with $r = [v]_M$.

- If $\varphi = tt$ or $\varphi = ff$ then there is no move neither from n nor $\overline{n}$.
- If $\varphi = q$ then there is a move from n to $(p, \Delta(q), v)$ and there is a move from $\overline{n}$ to $(p, \Delta(q), r)$.
- If $\varphi = \varphi_1 \vee \varphi_2$, and the player *Eve* moves to (p, φ_i, v), then in $\hat{\mathcal{G}}(\mathcal{P}, \mathcal{A}, M)$, she can move to (p, φ_i, r) with $i \in \{1, 2\}$ and conversely.
- If $\varphi = \varphi_1 \wedge \varphi_2$, and the player *Adam* moves to (p, φ_i, v) with $i \in \{1, 2\}$, then in $\hat{\mathcal{G}}(\mathcal{P}, \mathcal{A}, M)$, he can move to (p, φ_i, r) and conversely.
- Assume that $\varphi = \langle g \rangle \psi$.
  - Assume that the player *Eve* moves to (p, ψ, v'), for some $v' \in v\uparrow \cap [\![g]\!]$. Let $r' = [v']_M$. From Proposition 61, we get that if $M \geq M_{\mathcal{A}} \geq M_g$, then $r' \subseteq g$ and by definition $r' \in r\uparrow$. *Eve* can move to (p, ψ, r').
  - Reciprocally, if *Eve* moves to (p, ψ, r'), then $r' \subseteq g$. Let $v \in r$, according to Proposition 61, there is $v' \in v\uparrow \cap r'$. Since $r' \subseteq g$, then $v' \in [\![g]\!]$ and the player *Eve* can move to (p, ψ, v').
- The case when $\varphi = [g]\psi$ is obvious from the previous one.
- Assume that $\varphi = \langle a \rangle \psi$,
  - Assume that *Eve* moves to (p', ψ, v'), then $v' = v[h_a := 0]$ and there is $p \xrightarrow{g,a} p'$ with $v \in [\![g]\!]$. Let $r = [v]_M$. From Proposition 61, we get that if $M \geq M_{\mathcal{P}} \geq M_g$, then $r \subseteq g$. *Eve* can move to $(p', \psi, r[h_a := 0])$.
  - Conversely, if *Eve* moves to (p', ψ, r'), then $r' = r[h_a := 0]$ and there is $p \xrightarrow{g,a} p'$ such that $r \subseteq g$. Let $v \in r$, then $v \in [\![g]\!]$ and *Eve* can move to (p', ψ, v') with $v' = v[h_a := 0]$. By Proposition 61, $v' \in r'$.
- The case when $\varphi = [a]\psi$ becomes obvious.

□

From Proposition 147, if we want to check whether a timed process is accepted by a modal automaton, we check the existence of a winning strategy in the symbolic game with the suitable M parameter.

In consequence of Proposition 147 and Theorem 11, we can state the following result.

Theorem 148 *It is decidable whether a modal automaton $\mathcal{A}$ accepts a timed process $\mathcal{P}$.*

5.1.3 Restricted Modal Automata: WG-MA and C-MA

We define two subclasses of modal automata that intend to be equivalent to WG-WT$_\mu$ formulas, and C-WT$_\mu$ formulas (see Subsection 4.1.3). We call these automata well guarded automata (WG-MA for short) and modal automata for control (C-MA for short). These automata use *well guarded modal formulas* and *modal formulas for control* in their transition relations. The definition of modal formulas and well guarded modal formulas use discrete modal formulas.

Definition 149 Let S be a set. The set of discrete modal formulas over (Σ,S) is defined by the following rules:

- $\langle a\rangle s$ and $[a]s$ are discrete modal formulas provided that $a \in \Sigma$ and $s \in S$.
- $\varphi_1 \wedge \varphi_2$ and $\varphi_1 \vee \varphi_2$ is a discrete modal formulas provided that φ_1 and φ_2 are discrete modal formulas.

Definition 150 Let S be a set. The set of *well guarded modal formulas* over (Σ, S), $\mathcal{MF}_{wg}(\Sigma, S)$ is defined by the following recursive set of rules:

- tt, ff and s are well guarded modal formulas where $s \in S$.
- $\langle g\rangle\varphi_e$ and $[g]\varphi_e$ are well guarded modal formulas provided that φ_e is a discrete modal formula over $(\Sigma, \mathcal{MF}_{wg}(\Sigma, S))$ and $g \in Gds_\Sigma$.
- $\varphi_1 \wedge \varphi_2$ and $\varphi_1 \vee \varphi_2$ are well guarded modal formulas provided that $\varphi_1, \varphi_2 \in \mathcal{MF}_{wg}(\Sigma, S)$.

Definition 151 A *well guarded modal automaton* (WG-MA) is a modal automaton transition relation of which uses formulas of $\mathcal{MF}_{wg}(\Sigma, Q)$ where Q is the set of states of the automaton.

Definition 152 The set $\mathcal{MF}_c(\Sigma, S)$ of *modal formulas for control* over a set S is defined like $\mathcal{MF}_{wg}(\Sigma, S)$, but we require that for every formula of the form $\langle g\rangle\varphi$, the formula φ is a boolean combination of formulas of the form $\langle a\rangle\psi$, with $\psi \in \mathcal{MF}_c(\Sigma, S)$.

Definition 153 A *modal automaton for control*(C-MA) is a modal automaton transition relation of which uses formulas of $\mathcal{MF}_c(\Sigma, Q)$ where Q is the set of states of the automaton.

5.2 Automata and Logic

Now, we consider the relation between modal automata and WT$_\mu$ formulas. We show how to translate formulas of WT$_\mu$ into equivalent modal automata and vice versa. Such kind of transformation has been considered earlier for temporal [Var96, Var07] and modal logics [SE89, Wal95]. The proof of the translation is similar to the proof of a similar result [Wal01] in the setting of the μ-calculus. Assuming that ξ is one of $\{WG, C\}$, at the end of this section, we show that ξ-WT$_\mu$ formulas can be translated into an equivalent ξ-MA and, conversely. As we have proved that the satisfiability problem of C-WT$_\mu$ is decidable, so will be the emptiness problem of C-MA.

5.2.1 From Formulas to Modal Automata

Now we give the construction of a modal automaton $\mathcal{A}_\varphi$ whose language is the set of timed processes satisfying the formula φ. W.l.o.g, we assume that φ does not contain sub formulas of one of the forms tt, ff (indeed, the occurrence of such kinds of formulas in φ can be easily replaced with fixpoint formulas without changing the meaning of φ).

A state in $\mathcal{A}_\varphi$ corresponds to a sub formula of φ that we aim at verifying in a current state of a $(\mathcal{V}_\Sigma \cup \Sigma)$ labelled-transition system. The following clauses present how to reduce a local verification of a formula into local verifications of its sub formulas:

- To verify that $\varphi_1 \wedge \varphi_2$ in $sub(\varphi)$, we check φ_1 and φ_2 in the current state.
- To verify that $\varphi_1 \vee \varphi_2$ in $sub(\varphi)$, we check in a non deterministic way φ_1 or φ_2 in the current state.
- To verify that $\langle g \rangle \psi$ in $sub(\varphi)$, we check the existence of a successor of the current valuation of the clocks which satisfies g, then we check ψ in the current state of the transition system with respect to the new values of the clocks.
- To verify that $[g]\psi$ in $sub(\varphi)$, we check the existence of an eventual successor of the current valuation of the clocks (time elapse) which satisfies g, then we check ψ in the current state of the transition system with respect to the new values of the clocks.
- To verify that $\langle a \rangle \psi$ in $sub(\varphi)$, we check the existence of an a-successor of the current state of the transition system. Then we check ψ on that successor which became the current state.
- To verify that $[a]\psi$ in $sub(\varphi)$, we check the existence of an eventual a-successor of the current state of the transition system. Then we check ψ on that successor, which becomes the current state.
- To verify that $\sigma X.\varphi(X)$ in $sub(\varphi)$, we check $\varphi(X)$ in the current state with respect to the current valuation.
- To verify that X in $sub(\varphi)$, we check $\varphi(X)$ in the current state, where $\mathcal{D}(X) = \sigma X.\varphi(X)$.

Finally, in the construction of the automaton we must ensure constraints on fixpoints. That is, every μ-variable Y is infinitely often regenerated only when there ν-variable X that is greater that Y and infinitely often regenerated.

We will now give the transformation from a formula into an equivalent automaton. First we define a function tr that transform a WT_μ formula into a modal formula. The symbol $Var(\varphi)$ denotes the set of variables that occur in φ.

Definition 154 Given a WT_μ formula φ, the transition relation associated to φ is the modal formula $tr(\varphi) \in \mathcal{MF}(\Sigma, Var(\varphi))$ defined inductively as follows:

- $tr(\varphi_1 \wedge \varphi_2) = tr(\varphi_1) \wedge tr(\varphi_2)$
- $tr(\varphi_1 \vee \varphi_2) = tr(\varphi_1) \vee tr(\varphi_2)$
- $tr(\langle g \rangle \psi) = \langle g \rangle tr(\psi)$
- $tr([g]\psi) = [g]tr(\psi)$
- $tr(\langle a \rangle \psi) = \langle a \rangle tr(\psi)$
- $tr([a]\psi) = [a]tr(\psi)$
- $tr(\sigma X.\psi(X)) = X$
- $tr(X) = X$

Remark: By construction, it is not difficult to remark that, if φ is a formula of WG-WT_μ, then $tr(\varphi)$ is a modal formula of $\mathcal{MF}_{wg}(\Sigma, Var(\varphi))$; and if φ is a formula of C-WT_μ, then $tr(\varphi)$ is a modal formula of $\mathcal{MF}_c(\Sigma, Var(\varphi))$

The transformation of a formula into an equivalent automaton is the following.

Definition 155 For a formula φ, consider the automaton

$$\mathcal{A}_\varphi = \langle Q, \Sigma, q^0, \Delta_\mathcal{A} : Q \to \mathcal{MF}(\Sigma, Q), Acc \subseteq Q^\omega \rangle$$

where,

- $Q = Var(\varphi) \cup \{q^0\}$,
- q^0 is the initial state of the automaton.
- The transition relation $\Delta : Q \to \mathcal{MF}(\Sigma, Q)$ is defined by:
  - $\Delta(q^0) = tr(\varphi)$
  - if $q = X$ then $\Delta(q) = tr(\psi(X))$ where $\mathcal{D}_\varphi(X) = \sigma X.\psi(X)$,
- The acceptance condition is the parity condition that uses the parity function $rank : Q \to \mathbb{N}$ defined by:

$$rank(q) = \begin{cases} 0 & \text{if } q = q^0 \\ 2 \times alt(\mathcal{D}_\varphi(X)) & \text{where } q = X \text{ and } X \text{ is a } \nu\text{-variable} \\ 2 \times alt(\mathcal{D}_\varphi(X)) + 1 & \text{where } q = X \text{ and } X \text{ is a } \mu\text{-variable} \end{cases}$$

The alternation depth $alt(\varphi)$ of a WT$_\mu$ formula is defined like the alternation depth of μ-calculus formulas (see Definition 22). We also remark that the initial state q^0 never occurs in the transition relations of the states.

Lemma 156 Let φ be a formula and let $\mathcal{A}_\varphi$ be the automaton obtained from φ using the transformation above. If φ is a formula of of WG-WT$_\mu$ (resp. C-WT$_\mu$) then $\mathcal{A}_\varphi$ is a WG-MA (resp. C-MA).

Proof
If $q = q^0$ then $\Delta(q) = tr(\varphi)$. According to the remark above, if φ is a formula of WG-WT$_\mu$ (resp. C-WT$_\mu$) then $tr(\varphi) \in \mathcal{MF}_{wg}(\Sigma, Q)$ (resp. $tr(\varphi) \in \mathcal{MF}_c(\Sigma, Q)$.
If $q = X$ is a variable, then $\Delta(q) = \Delta(\psi(X))$ where, $\mathcal{D}_\varphi(X) = \psi(X)$. If φ is a formula of WG-WT$_\mu$ (resp. C-WT$_\mu$) then, by definition, $\psi(X)$ is also a formula of WG-WT$_\mu$ (resp. C-WT$_\mu$). We use the remark above to conclude. □

Theorem 157 *Given a formula φ, for every timed process $\mathcal{P}$,*

$$\mathcal{P} \vDash \varphi \text{ if and only if } \mathcal{P} \in \mathcal{L}(\mathcal{A}_\varphi)$$

Proof

($\Longrightarrow$) We show that if $\mathcal{P} \vDash \varphi$, then $\mathcal{P} \in \mathcal{L}(\mathcal{A}_\varphi)$. For this we show the existence of a winning strategy for the player *Eve* in $\mathcal{G}(\mathcal{P}, \mathcal{A}_\varphi)$. Recall that in $\mathcal{G}(\mathcal{P}, \mathcal{A}_\varphi)$, the player *Eve* makes a choice in positions of the form $(p, \varphi_1 \vee \varphi_2, v)$ or $(p, \langle g \rangle \psi, v)$ or $(p, \langle a \rangle \psi, v)$. The choice should be done as follows:

- In a position $(p, \varphi_1 \vee \varphi_2, v)$ he should choose (p, φ_i, v) with $i \in \{1, 2\}$ and ${}^\mu\mathtt{sig}((p, v), \varphi_1 \vee \varphi_2) = {}^\mu\mathtt{sig}((p, v), \varphi_i)$.
- In a position $(p, \langle g \rangle \psi, v)$ he should choose the (p, ψ, v') with $v' \in v\uparrow$, $v' \in [\![g]\!]$ and ${}^\mu\mathtt{sig}((p, v), \langle g \rangle \psi) = {}^\mu\mathtt{sig}((p, v'), \psi)$.
- In a position $(p, \langle a \rangle \psi, v)$ he should choose the (p, ψ, v') with ${}^\mu\mathtt{sig}((p, v), \langle a \rangle \psi) = {}^\mu\mathtt{sig}((p, v'), \psi)$ $v' = v[h_a := 0]$, and p' is such that there is $p \xrightarrow{g,a} p'$ and $v \in [\![g]\!]$.

We show that such a strategy is winning for the player *Eve*. Indeed, assume that there is a play $(p_1, \varphi_1, v_1)(p_2, \varphi_2, v_2) \ldots$ consistent with the above strategy on which some odd priority p is the greatest priority appearing infinitely often. This means that on this play we infinitely often meet the μ-variable X_l where $l = (p-1)/2$. Let m be a step of the play after which no priority greater than p appears. In particular it means that after m there are no variables with indexes greater than l. By the signature decrease lemma (see Lemma 138), the signatures of positions of the play after m never increase on the first l positions. They decrease every time we meet X_l. But this is impossible as the lexicographic order on l-tuples of ordinals is well ordering. Hence, such a play can not exist, and the strategy we have defined is winning for player *Adam*.

($\Longleftarrow$) In this direction of the proof, we show that if $\mathcal{P} \in \mathcal{L}(\mathcal{A}_\varphi)$, then $\mathcal{P} \vDash \varphi$. We assume that $\mathcal{P} \in \mathcal{L}(\mathcal{A}_\varphi)$ and $\mathcal{P} \nvDash \varphi$, then we exhibit a winning strategy for the player *Adam* in $\mathcal{G}(\mathcal{P}, \mathcal{A}_\varphi)$. If $\mathcal{P} \nvDash \varphi$, the strategy for the player *Adam* is dual to the one of the player *Eve* stated in the previous direction. It works as follows:

- In a position $(p, \varphi_1 \wedge \varphi_2, v)$ he should choose (p, φ_i, v) with $i \in \{1, 2\}$ and ${}^{\nu}\mathtt{sig}((p, v), \varphi_1 \wedge \varphi_2) = {}^{\nu}\mathtt{sig}((p, v), \varphi_i)$.
- In a position $(p, [g]\psi, v)$ he should choose the (p, ψ, v') with $v' \in v\uparrow$, $v' \in [\![g]\!]$ and ${}^{\nu}\mathtt{sig}((p, v), [g]\psi) = {}^{\nu}\mathtt{sig}((p, v'), \psi)$.
- In a position $(p, [a]\psi, v)$ he should choose the (p, ψ, v') with ${}^{\nu}\mathtt{sig}((p, v), [a]\psi) = {}^{\nu}\mathtt{sig}((p, v'), \psi)$ $v' = v[h_a := 0]$, and p' is such that there is $p \xrightarrow{g,a} p'$ and $v \in [\![g]\!]$.

Using a similar argument as in the direction ($\Longrightarrow$), we show that there is not a μ-variable which is infinitely often regenerated in the game. Then, we get a contradiction with that $\mathcal{P} \in \mathcal{L}(\mathcal{A}_\varphi)$.

□

5.2.2 From Modal Automata to Formulas

This transformation is similar to the transformation in [Wal01] for the case of the μ-calculus and it uses vectorial formulas (see [Bek84, AN01]).

Definition 158 A system of equations of WT_μ is a system:

$$\begin{array}{lcl} X_1 & \stackrel{\sigma_1}{=} & \varphi_1(X_1, \cdots, X_n) \\ X_2 & \stackrel{\sigma_2}{=} & \varphi_2(X_1, \cdots, X_n) \\ \vdots & & \\ X_n & \stackrel{\sigma_n}{=} & \varphi_n(X_1, \cdots, X_n) \end{array}$$

where for every $i \in \{1, \cdots, n\}$, σ_i is a fixpoint operator and φ_i is an WT_μ formula that does not have a fixpoint sub formula.

The solution of a system of n equations is a vector of n formulas of WT_μ defined by induction on n as follows:

- The solution of a system made of a unique equation $X_1 \stackrel{\sigma_1}{=} \varphi_1(X_1)$ is the formula $\sigma_1 X.\varphi_1(X_1)$
- The solution of a system of n equations:

  $$\begin{array}{lcl} X_1 & \stackrel{\sigma_1}{=} & \varphi_1(X_1, \cdots, X_{n-1}, X_n) \\ X_2 & \stackrel{\sigma_2}{=} & \varphi_2(X_1, \cdots, X_{n-1}, X_n) \\ \vdots & & \\ X_n & \stackrel{\sigma_n}{=} & \varphi_n(X_1, \cdots, X_{n-1}, X_n) \end{array}$$

  is the vector $(\psi_1, \cdots, \psi_{n-1}, \sigma_n X_n.\varphi_n(\psi_1, \cdots, \psi_{n-1}, X_n))$ where $(\psi_1, \cdots, \psi_{n-1})$ is the solution of the system of $n-1$ equations obtained by replacing X_n with the formula $\sigma_n X_n.\varphi_n(X_1, \cdots, X_{n-1}, X_n)$:

  $$\begin{array}{lcl} X_1 & \stackrel{\sigma_1}{=} & \varphi_1(X_1, \cdots, X_{n-1}, \sigma_n X_n.\varphi_n(X_1, \cdots, X_{n-1}, X_n)) \\ X_2 & \stackrel{\sigma_2}{=} & \varphi_2(X_1, \cdots, X_{n-1}, \sigma_n X_n.\varphi_n(X_1, \cdots, X_{n-1}, X_n)) \\ \vdots & & \\ X_{n-1} & \stackrel{\sigma_{n-1}}{=} & \varphi_{n-1}(X_1, \cdots, X_{n-1}, \sigma_n X_n.\varphi_n(X_1, \cdots, X_{n-1}, X_n)) \end{array}$$

The use of system of equations do not add the expressive power of WT$_\mu$; indeed, every formula φ of WT$_\mu$ can be defined as the first component of the solution of the system of equations in which each equation $X \stackrel{\sigma}{=} \psi(X)$ corresponds to a sub formula $\sigma X.\psi(X)$ of φ and the order on the equations depends on the dependency order between variables in φ.

In what follows, we give the transformation of an automaton into an equivalent formula. The resulting formula is the component of a system of equations such that each equation corresponds to a unique state of the modal automaton. We use the parity indexes to define the order between two equations.

Definition 159 Take an automaton

$$\mathcal{A} = \langle Q, \Sigma, q^0, \Delta_{\mathcal{A}} : Q \to \mathcal{F}(\Sigma, Q), Acc \subseteq Q^\omega \rangle$$

and $q_1, \cdots, q_n$ an order over the states of $\mathcal{A}$ such that for $i < j$, we have $rank(q_i) \geq rank(q_j)$. If the initial state of $\mathcal{A}$ is q_k according to the order above, we define the formula $\varphi_{\mathcal{A}}$ as the k^{th} component of the solution of the following system of equations:

$$\begin{array}{lcl} X_1 & \stackrel{\sigma_1}{=} & \varphi_1(X_1, \cdots, X_{n-1}, X_n) \\ X_2 & \stackrel{\sigma_2}{=} & \varphi_2(X_1, \cdots, X_{n-1}, X_n) \\ \vdots & & \\ X_n & \stackrel{\sigma_n}{=} & \varphi_n(X_1, \cdots, X_{n-1}, X_n) \end{array}$$

X_i is the variable associated to q_i and $\varphi_i(X_1, \cdots, X_{n-1}, X_n)$ is obtained from $\Delta(q_i)$ by replacing each state by the corresponding variable. We put $\sigma_i = \mu$ if $rank(q_i)$ is odd and $\sigma_i = \nu$ if $rank(q_i)$ is even.

The proof of the following theorem is similar to the proof of Theorem 157; it also uses signature decrease lemma (see Lemma 138).

Theorem 160 *Given an automaton $\mathcal{A}$, for every timed process $\mathcal{P}$,*

$$\mathcal{P} \in \mathcal{L}(\mathcal{A}) \text{ if and only if } \mathcal{P} \vDash \varphi_{\mathcal{A}}$$

The following corollary is a consequence of Lemma 156, Theorem 157 and, Theorem 160.

Corollary 161 Every WG-WT$_\mu$ formula can be translated into an equivalent WG-MA and conversely. Every C-WT$_\mu$ formula can be translated into an equivalent C-MA and conversely.

In consequence of Corollary 161 and the result in Theorem 160, we get the decidability of the emptiness problem of C-MA which is to check whether there exists a timed process process $\mathcal{P}$ that satisfies a given C-MA.

Theorem 162 *There is decision procedure that checks whether a C-MA is empty. Moreover, if a C-MA is not empty, we can construct one of its models.*

5.3 Quotient for Automata

We consider WG-MA and we define the *quotient* $\mathcal{A}/\mathcal{P}$ of a WG-MA $\mathcal{A}$ over a timed process $\mathcal{P}$ that satisfies the following property:

$$\mathcal{R} \times \mathcal{P} \vDash \mathcal{A} \text{ if and only if } \mathcal{R} \vDash \mathcal{A}/\mathcal{P}.$$

We show that the quotient of a C-MA over a timed process is still a C-MA. Later in Section 5.4, we use this quotient to provide a solution to controller synthesis problems.

Definition 163 Given a WG-MA

$$\mathcal{A} = \langle Q, \Sigma, q^0, \Delta : Q \to \mathcal{MF}(\Sigma, Q), Acc \subseteq Q^\omega \rangle$$

and a timed process

$$\mathcal{P} = \langle P, \Sigma, p^0, \Delta_P \rangle$$

we construct the WG-MA $\mathcal{A}/\mathcal{P}$.

Firstly, we propose the division φ/p of a formula $\varphi \in \mathcal{MF}_{wg}(\Sigma, Q)$ by a state $p \in P$. Let M be the biggest constant used in $\mathcal{P}$. We assume that φ is M-rectangular. The result of the division is a well guarded modal formula from $\mathcal{MF}_{wg}(\Sigma, Q \times P)$ as stated below:

$$\begin{aligned}
tt/p &= tt \\
ff/p &= ff \\
q/p &= (q, p) \\
(\varphi \vee \psi)/p &= (\varphi/p) \vee (\psi/p) \\
(\varphi \wedge \psi)/p &= (\varphi/p) \wedge (\psi/p) \\
([g]\varphi)/p &= [g](\varphi/(p, g)) \\
(\langle g \rangle \varphi)/p &= \langle g \rangle (\varphi/(p, g)) \\
(\langle a \rangle \varphi)/(p, g) &= \langle a \rangle \left(\bigvee_{p \xrightarrow{g,a} p'} (\varphi/p') \right) \\
([a]\varphi)/(p, g) &= [a] \left(\bigwedge_{p \xrightarrow{g,a} p'} (\varphi/p') \right)
\end{aligned}$$

Given two discrete modal formulas φ and ψ, we define $(\varphi \vee \psi)/(p, g) = \varphi/(p, g) \vee \psi/(p, g)$ and $(\varphi \wedge \psi)/(p, g) = \varphi/(p, g) \wedge \psi/(p, g)$.

By convention, a disjunction over an empty set is false, and a conjunction over an empty set is true.

Finally, we define the *quotient*,

$$\mathcal{A}/\mathcal{P} = \langle Q \times P, \Sigma, (q^0, p^0), \Delta_/ : Q \times P \to \mathcal{MF}(\Sigma, Q), Acc \subseteq Q^\omega \rangle$$

where $\Delta_/(q, p) = \Delta(q)/p$.

Lemma 164 Let $\varphi \in \mathcal{MF}_c(\Sigma, S)$ and let $p \in P$ be a state of a timed process. The quotient φ/p is a formula of $\mathcal{MF}_c(\Sigma, S)$.

Proof
The proof uses the induction principle, on the size of the formula. All the cases but when φ is in one of the forms $[g]\psi$ or $\langle g\rangle\psi$ are obvious.

- if $\varphi = \langle g\rangle\psi$, then ψ is a boolean combination of the form $\langle a_1\rangle\psi_1 \bowtie_1 \ldots \bowtie_{n-1} \langle a_n\rangle\psi_n$ where $\bowtie_i \in \{\vee, \wedge\}$ and all ψ_i are in $\mathcal{MF}_c(\Sigma, Q)$.
  Now we show that φ/p belongs to $\mathcal{MF}_c(\Sigma, Q \times P)$. Obviously, $\varphi/p = \langle g\rangle((\langle a_1\rangle\psi_1 \bowtie_1 \ldots \bowtie_{n-1} \langle a_n\rangle\psi_n)/(p,g))$ which is equal to $\langle g\rangle((\langle a_1\rangle\psi_1)/(p,g) \bowtie_1 \ldots \bowtie_{n-1} (\langle a_n\rangle\psi_n)/(p,g))$. We show that each member of the combination has the appropriate form. By definition $(\langle a_i\rangle\psi_i)/(p,g) = \langle a_i\rangle(\bigvee_{p \xrightarrow{g,a} p'}(\psi_i/p'))$. We remark that $(\langle a_i\rangle\psi_i)/(p,g)$ may be equal to $\langle a_i\rangle ff$ if there is no p' such that $p \xrightarrow{g,a} p'$.
  As by induction hypothesis $\psi_i/p' \in \mathcal{MF}_c(\Sigma, S)$, it follows that φ/p belongs to $\mathcal{MF}_c(\Sigma, Q \times P)$ as each formula participating in the boolean combination after $\langle g\rangle$ is the of form $\langle a\rangle\psi$.

- if $\varphi = [g]\psi$, then ψ is a boolean combination of the form $\langle a_1\rangle\psi_1 \bowtie_1 \ldots \bowtie_{n-1} \langle a_n\rangle\psi_n \bowtie [a_{n+1}]\psi_{n+1} \bowtie_{n+1} \ldots \bowtie_{n+m-1} [a_{n+m}]\psi_{n+m}$ where $\bowtie_i \in \{\vee, \wedge\}$ and ψ_i is in $\mathcal{MF}_c(\Sigma, Q)$ with $(i = 1..n+m)$.
  Now we show that φ/p belongs to $\mathcal{MF}_c(\Sigma, Q \times P)$. Obviously, $\varphi/p = [g]((\langle a_1\rangle\psi_1)/(p,g) \bowtie_1 \ldots \bowtie_{n-1} (\langle a_n\rangle\psi_n)/(p,g) \bowtie_n ([a_{n+1}]\psi_{n+1})/(p,g) \bowtie_{n+1} \ldots \bowtie_{n+m-1} ([a_{n+m}]\psi_{n+m})/(p,g))$. We need to show that each member of the combination has the appropriate form. We consider the following two cases:

  - By definition $(\langle a_i\rangle\psi_i)/(p,g) = \langle a_i\rangle(\bigvee_{p \xrightarrow{g,a} p'}(\psi_i/p'))$. We remark that $(\langle a_i\rangle\psi_i)/(p,g)$ may be equal to $\langle a_i\rangle ff$ if there is no p' such that $p \xrightarrow{g,a} p'$.
  - By definition $([a_i]\psi_i)/(p,g) = [a_i](\bigwedge_{p \xrightarrow{g,a} p'}(\psi_i/p'))$. We remark that $([a_i]\psi_i)/(p,g)$ may be equal to $[a_i]tt$ if there is no p' such that $p \xrightarrow{g,a} p'$.

  As by induction hypothesis $\psi_i/p' \in \mathcal{MF}_c(\Sigma, S)$, it follows from the two cases just above that φ/p belongs to $\mathcal{MF}_c(\Sigma, Q \times P)$.

□

In consequence of Lemma 164, we get the following corollary.

Corollary 165 The quotient of a C-MA by a timed process is a C-MA.

Now, we show the main property of the quotient operator.

Theorem 166 *Let $\mathcal{P}$ be a timed process and $\mathcal{A}$ be a modal automaton, both over an alphabet Σ. The modal automaton $\mathcal{A}/\mathcal{P}$ is such that for every timed process $\mathcal{R}$ over Σ :*

$$\mathcal{P} \times \mathcal{R} \vDash \mathcal{A} \quad \textit{if and only if} \quad \mathcal{R} \vDash \mathcal{A}/\mathcal{P}$$

Proof
We will consider $\mathcal{G}(\mathcal{R} \times \mathcal{P}, \mathcal{A}) = \langle N_E, N_A, T, Acc \rangle$ and $\mathcal{G}(\mathcal{R}, \mathcal{A}/\mathcal{P}) = \langle \overline{N}_E, \overline{N}_A, \overline{T}, Acc' \rangle$. We say that a position $n = ((s,p), \varphi, v)$ of the game $\mathcal{G}(\mathcal{R} \times \mathcal{P}, \mathcal{A})$ is *relevant* if $\varphi \in \mathcal{MF}(\Sigma, Q)$. We write $n \rightarrow_* n'$ if n and n' are positions and there is a path from n to n'.

Let us define the map f from the positions of $\mathcal{G}(\mathcal{R} \times \mathcal{P}, \mathcal{A})$ to those of $\mathcal{G}(\mathcal{R}, \mathcal{A}/\mathcal{P})$. To a position $((s,p), \varphi, v)$ we associate $(s, \varphi/p, v)$ with one exception when φ starts with modality $\langle a \rangle$ or $[a]$, for some action a. In this later case we associate to $((s,p), \varphi, v)$ the node $(s, \varphi/(p,g), v)$ where g is the unique atomic constraint such that $v \vDash g$.

Let us take a pair of positions n and $f(n)$. We will show how a move of a player from n to some n' can be mimicked by moves of the same player from $f(n)$ to $f(n')$. Similarly, we will show that a sequence of moves from $f(n)$ to some $f(n')$ can be mimicked by a move from n to n'.

The proof is easy for all but positions with formulas starting with an action modality. Let us consider several cases:

- Suppose $n = ((s,p), \langle a \rangle \varphi, v)$. Then $f(n) = (s, (\langle a \rangle \varphi)/(p,g), v)$ where g is the unique atomic constraint such that $v \vDash g$. From n Eve can go to a position $((s', p'), \varphi, v[h_a := 0])$; where $s \xrightarrow{g_s, a} s'$ and $p \xrightarrow{g_p, a} p'$ with $v \vDash g_s$ and $v \vDash g_p$. Observe that $g_p = g$ as g_p is an atomic constraint. By definition

$$(\langle a \rangle \varphi)/(p,g) = \langle a \rangle \left(\bigvee_{p \xrightarrow{g,a} p_1} (\varphi/p_1) \right)$$

  This means that from $f(n)$ Eve can go to $(s', \bigvee_{p \xrightarrow{g,a} p'} (\varphi/p'), v[h_a := 0])$. By choosing the disjunct with p' she can get to $f(n') = (s', \varphi/p', v[h_a := 0])$.

- Let us take n and n' as above and show that every choice of Eve from $f(n)$ can be mimicked from n. By this we mean that after making two moves from $f(n)$ Eve has to hit a position of the form $f(n')$ for some n' and we will show that Eve can reach n' from n.

  As $f(n) = (s, (\langle a \rangle \varphi)/(p,g), v)$, Eve can move to

$$\left(s', \bigvee_{p \xrightarrow{g,a} p_1} (\varphi/p_1), v[h_a := 0] \right)$$

  where $s \xrightarrow{g_s, a} s'$ and $v \vDash g_s$. Then Eve can chose one of the disjuncts and get to $(s', \varphi/p', v[h_a := 0])$. Clearly this node is of the form $f(n')$ for $n' = ((s', p'), \varphi, v[h_a := 0])$. Since $v \vDash g$ we have that from $n = ((s,p), \langle a \rangle \varphi, v)$ Eve can get to $n' = ((s', p', \varphi, v[h_a := 0])$ as required.

We remark that the case when $n = ((s,p), [a]\varphi, v)$ is dual to the case above. We have shown that a move in $\mathcal{G}(\mathcal{P} \times \mathcal{R}, \mathcal{A})$ can be mimicked by a unique sequence of moves in $\mathcal{G}(\mathcal{R}, \mathcal{A}/\mathcal{P})$ and reciprocally. We have also shown that the set of states of $\mathcal{A}$ occurring in a move of $\mathcal{G}(\mathcal{R} \times \mathcal{P}, \mathcal{A})$ is the same as the set of states of $\mathcal{A}$ occurring in $\mathcal{G}(\mathcal{R}, \mathcal{A}/\mathcal{P})$. As the winning

condition in $\mathcal{G}(\mathcal{R} \times \mathcal{P}, \mathcal{A})$ and $\mathcal{G}(\mathcal{R}, \mathcal{A}/\mathcal{P})$ depends on states of $\mathcal{A}$ encountered along a play, we get that there a play is winning for some in $\mathcal{G}(\mathcal{P} \times \mathcal{R}, \mathcal{A})$ if and only if the mimicked play is winning for the same player in $\mathcal{G}(\mathcal{R}, \mathcal{A}/\mathcal{P})$. Then we get that $\mathcal{A}$ accepts $\mathcal{R} \times \mathcal{P}$ if and only if $\mathcal{A}/\mathcal{P}$ accepts $\mathcal{R}$. □

5.4 Centralised Controller Synthesis for C-WT$_\mu$

We use the quotient operation for two controller synthesis problems. In Theorem 162, we have established the decidability of the emptiness of C-MA. Moreover we are able to construct models for a non empty C-MA. We use that result to provide solutions to the centralised controller synthesis problem and Δ-dense-time control problem. Controllers will be models of quotients of specifications over plants. Controllers will also satisfy additional properties. An additional property is for example a control condition (hypothesis) that we describe with C-MA.

5.4.1 Centralised Controller Synthesis

We assume that Σ, the set of events is partitioned into the set Σ_u of *uncontrollable* events and, the set Σ_c of *controllable* events; in the other words $\Sigma = \Sigma_u \cup \Sigma_c$ with $\Sigma_u \cap \Sigma_c = \emptyset$.

The *centralized control problem(CCP)* we consider is the following:

CCP: Given $\Sigma = \Sigma_u \cup \Sigma_c$, a timed process $\mathcal{P}$ and a C-MA over Σ, does there exists a timed process $\mathcal{R}$ over Σ, satisfying the *control condition* (CC), such that $\mathcal{P} \times \mathcal{R} \vDash \mathcal{A}$.

Control condition (CC): $\mathcal{R}$ does not forbid occurrences of uncontrollable events.

The condition (CC) needs some attention and a construction. We need to describe a property that prevents $\mathcal{R}$ from restricting the occurrence of an uncontrollable event at any moment of time; we also need to describe a property that makes it possible for $\mathcal{R}$ to forbid the occurrence of a controllable event in some time instances.
We claim that these two properties for $\mathcal{R}$ can be described with the C-modal automaton $\mathcal{B}$ defined as follows:

$$\mathcal{B} = \langle \{q^0\}, \Sigma, q^0, \Delta_{\mathcal{B}} : Q \to \mathcal{MF}_c(\Sigma, Q), Acc = (q^0)^\omega \rangle$$

where,

$$\Delta_{\mathcal{B}}(q^0) = \left(\bigwedge_{a \in \Sigma_u} [tt]\langle a \rangle q^0 \right) \wedge \left(\bigwedge_{a \in \Sigma_c} [tt][a] q^0 \right)$$

Proposition 167 $\mathcal{R} \vDash \mathcal{B}$ if and only if $\mathcal{R}$ satisfies the control condition (CC) above.

Proof
Let $\mathcal{R} = \langle S, \Sigma, s^0, \Delta_{\mathcal{R}} \rangle$ be a timed process such that $\mathcal{R} \vDash \mathcal{B}$; then there is a winning strategy for the player *Eve* in the acceptance game $\mathcal{G}(\mathcal{R}, \mathcal{B})$. We will show that in every state of $\mathcal{R}$ any

uncontrollable event can happen at any time and some controllable events may not happen at some time instance.

A play in $\mathcal{G}(\mathcal{R},\mathcal{B})$ starts in (s^0,q^0,v^0) which is a position for *Adam*, but a winning position for *Eve*. Assume that a play is in a winning position (s,q^0,v). There is two cases:

1. *Adam* moves to $(s,\bigwedge_{a\in\Sigma_u}([tt]\langle a\rangle q^0),v)$ which is position for *Adam*, then he can move to $(s,[tt]\langle a\rangle q^0,v)$ for any $a\in\Sigma_u$ and then move to $(s,\langle a\rangle q^0,v+t)$ for any $t\in\mathbb{R}^+$ as $v+t\in[\![tt]\!]$. The later position is a position for *Eve*. As there is a winning strategy from that later position, the player *Eve* can move to $(s',q^0,v+t[h_a:=0])$ meaning that, there is a transition $s\xrightarrow{g,a}s'$ with $v+t\in[\![g]\!]$. Obviously, $\mathcal{R}$ does not prevent the occurrence of the event a at the time $v+t$ for any time $t\in\mathbb{R}^+$ and any event $a\in\mathcal{A}_u$. The position $(s',q^0,v+t[h_a:=0])$ is a position for *Adam* but also a winning position for *Eve* from which we can repeat the argument.

2. *Adam* moves to $(s,\bigwedge_{a\in\Sigma_c}([tt][a]q^0),v)$ which is a position of *Adam*, then he can move to $(s,[tt][a]q^0,v)$ for any $a\in\Sigma_u$ and then moves to $(s,[a]q^0,v+t)$ for any $t\in\mathbb{R}^+$ as $v+t\in[\![tt]\!]$. The later position is a position for *Adam* and there are two cases:

   (a) There is no move from that later position meaning that there is not transition $s\xrightarrow{g,a}s'$ with $v+t\in[\![g]\!]$. Obviously, the controller prevents the occurrence of the event a at the time $v+t$.

   (b) There is a move from that later position to some position $(s',q^0,v+t[h_a:=0])$, meaning that $\mathcal{R}$ does not prevent the event a at the time context $v+t$. The position $(s',q^0,v+t[h_a:=0])$ is a position for *Adam* but also a winning position for *Eve* from which we can apply a similar argument.

We have shown that a state s of $\mathcal{R}$ that occurs in a winning position in the acceptance game satisfies the condition (CC). Because from a winning position we always move to another winning position containing s or a successor of s, we get that every state of $\mathcal{R}$ satisfies the condition (CC). □

A solution the the controller synthesis problem is given by the following result

Theorem 168

$$\mathcal{R}\vDash(\mathcal{A}/\mathcal{P})\wedge\mathcal{B}\ \textit{if and only if}\ \begin{cases}\mathcal{P}\times\mathcal{R}\vDash\mathcal{A}\\ \mathcal{R}\vDash\mathcal{B}\end{cases}$$

Proof
$\mathcal{P}\times\mathcal{R}\vDash\mathcal{A}$ if and only if (see Theorem 166) $\mathcal{R}\vDash\mathcal{A}/\mathcal{P}$. From Proposition 146, $\mathcal{R}\vDash\mathcal{B}$ and $\mathcal{R}\vDash\mathcal{A}/\mathcal{P}$ if and only if $\mathcal{R}\vDash(\mathcal{A}/\mathcal{P})\wedge\mathcal{B}$ □

Corollary 169 The CCP problem is decidable.

5.4.2 The Δ-Dense-Time Control Problem

The centralised Δ-dense-time control amounts to finding a controller (also called a Δ-controller) for a system such that at least $\Delta\geq 0$ time units elapse between two consecutive controllable events. The Δ-control condition is the following.

Δ control condition (Δ-CC) The intervals between any two controllable events are greater or equal to Δ.

Our solution to the centralised Δ-dense-time control is to build an automaton, called the Δ-automata. A Δ-controller should also satisfy the control condition (CC) above.

Let $\mathcal{B}_\Delta$ be the C-MA defined as follows:

$$\mathcal{B}_\Delta = \langle Q, \Sigma, q^0, \delta_\Delta : Q \to \mathcal{MF}_c(\Sigma, Q), Acc \subseteq Q^\omega \rangle$$

where

- $Q = \{q^0\} \cup \bigcup_{a \in \Sigma_c} \{q_a\}$ is the set of states. The state q_a is the one associated to the event a.
- q^0 is the initial state and it is not associated to any event.
- The transition relation is defined by:

$$\delta_\Delta(q^0) = [tt] \bigwedge_{u \in \Sigma_u} \langle u \rangle q^0 \wedge [tt] \bigwedge_{a \in \Sigma_c} [a] q_a$$

  and for every a the transition from the corresponding state q_a is defined by:

$$\delta_\Delta(q_a) = [tt] \bigwedge_{u \in \Sigma_u} \langle u \rangle q_a \wedge \left(\left([h_a < \Delta] \bigwedge_{b \in \Sigma_c} [b] ff \right) \vee \left([h_a \geq \Delta] \bigwedge_{b \in \Sigma_c} [b] q_b \right) \right)$$

- Acc is the parity condition defined with a function $rank$ which assigns the value 0 to every state in Q.

Let us comment the modal automaton $\mathcal{B}_\Delta$. The automaton $\mathcal{B}_\Delta$ has $|\Sigma_c| + 1$ states. The initial state q^0 describes what happens in the controller when no controllable event has occurred. At this step, any uncontrollable event may happen whatever is the time instance and any controllable event may happen at any time context; this is because no controllable event has occurred. The other states, one per controllable event, enable to save the information on the last controllable event that has occurred. At a state q_a, we assume that a is the last event that has occurred. At q_a, an occurrence of an uncontrollable event, u can not be prevented whatever is the timing context. Additionally, a controllable event b may occur if the amount of time elapsed since the occurrence of a (recall that a is the last event that has occurred), measured with the clock h_a, is greater or equal to Δ.

We state that a timed process satisfying $\mathcal{B}_\Delta$ also satisfies the condition $\Delta-$(CC). The proof of this proposition is similar to the proof of Proposition 167.

Proposition 170 For any timed process $\mathcal{R}$, $\mathcal{R} \in \mathcal{L}(\mathcal{B}_\Delta)$ if and only if $\mathcal{R}$ satisfies the condition $\Delta-$(CC).

In corollary to Theorem 168, we get the decidability of the Δ-dense-time control.

Corollary 171 The Δ-dense-time control is decidable; moreover, we can build a Δ-controller.

5.5 Conclusion

In this chapter we have considered two controller synthesis problems for plants described by timed processes and specifications described by C-WT$_\mu$: the centralised controller synthesis problem and the Δ-dense-time control problem. As a plant is a reactive system, we have assumed that events can be controllable (event completed by the plant) or not (events completed by the environment of the plant) and, only the occurrences of controllable events can be disabled by controllers.
We have shown that these two problems are decidable and, controller can be constructed when plants are controllable.

Conclusion and Perspectives

The main goal of this thesis was to provide methods to synthesise controllers for a class of real-time systems and real-time control objectives. In recent years a framework for the supervisory control of untimed reactive systems have been developed, in particular the framework that uses Kripke structure models with the standard μ-calculus. We wanted to use a class of real-time models for systems that could provide a framework for the supervisory control of real-time systems and if possible, reuse techniques proposed for the setting of untimed reactive systems.

Relying on some theoretical results on event-recording automata including closure under boolean operations and closure under complementation, we have chosen to work with these models. To describe real-time control objectives, we have chosen Event-Recording Logic (ERL) because it is a decidable timed extension of the μ-calculus and there are significant results for the control of untimed reactive systems with the μ-calculus.

Following our intuitions, we were interested in the similarities between untimed models and our models. These similarities concern some basic problems including the reachability analysis, the model-checking, the satisfiability, the disjunctive normal form theorem, the existence of deterministic models for specifications. These basic problems have been fundamental for the solution to the supervisory control of untimed systems.

ERL is too weak Our intuitions were good as, we had provided new decision procedures for the model-checking and the satisfiability problems of ERL. These procedures are new and interesting in the way that they reuse decision procedures for the model-checking in the setting of the μ-calculus with Kripke structures. A great benefit of the similarity of the two aforementioned problems, is that they have allowed us to provide a disjunctive normal form theorem for ERL. We had at that time, some useful theoretical ingredients to apply the methods Arnold et al. [AVW03, ABPV05, AW07] for the supervisory control. Unfortunately, we have discovered that ERL is not expressive enough to describe interesting properties for the controllers like *"An uncontrollable event can be completed at every time"*.

Overcome the weakness of ERL: the new logic WT_μ We have introduced a new language that we have called WT_μ. The logic WT_μ is a weak real-time extension of the standard μ-calculus. Compared to ERL, WT_μ considers modality indexed with timing constraints and modalities indexed with events. We have shown that WT_μ is strictly more expressive than ERL. We have hoped that the modification of the contents of modalities will be without adverse consequences. For the model-checking problem of WT_μ, we have provided a decision procedure similar to a procedure for the model-checking problem of the (untimed) μ-calculus. We have presented a fragment for WT_μ called *WT_μ for control* (C-WT_μ). We have provided

a decision procedure for the satisfiability problem of C-WT_μ; this procedure does not assume a limit on the constants of the models, and it shows how to construct a witness model for a satisfiable formula.

Centralised Control Result Relying on satisfiability results on C-WT_μ, we have proposed a *quotient-based approach* to a centralised controller synthesis problem and a centralised Δ-dense time controller synthesis problem for the class of real-time systems we have considered.

Perspectives

This thesis makes a progress in the domain of the controller synthesis of real-time systems. Presented results do not cover the class of real-time systems that can be modelled with timed automata, but they can be useful for the class of systems that we have considered. We think that our contribution can be useful for some parts of automated cars systems [1] and protocols (for instance Philips audio protocol). The approach that has been proposed in this thesis should be followed to provide more general results including the decidability of the existence of deterministic models for C-WT_μ, the decidability of the satisfiability problem of WT_μ, and the decentralised controller synthesis with WT_μ. We think that, if one assumes a bound on the constants used by the controllers, it will not be very difficult to provide a solution to the decentralised controller synthesis with WT_μ (and WG-WT_μ). Works in that direction may follow some results for the setting of the μ-calculus [AW07]. We also hope that works in the aforementioned directions can enable a better understanding of real-time models including timed automata models, and the logic L_ν [BCL05].

[1] See for example the European Project CityMobil at http://www.citymobil-project.eu

Bibliography

[ABPV05] André Arnold, Xavier Briand, Gérald Point, and Aymeric Vincent. A generic approach to the control of discrete event systems. In *44th IEEE Conference on Decision and Control 2005 and 2005 European Control Conference (CDC-ECC'05)*, pages 1–5. IEEE Computer Society, december 2005.

[ACD+92] Rajeev Alur, Costas Courcoubetis, David L. Dill, Nicolas Halbwachs, and Howard Wong-Toi. An implementation of three algorithms for timing verification based on automata emptiness. In *Proceedings of the 13th Symposium on Real-Time Systems(RTS'92:)*, pages 157–166. IEEE Computer Society Press, december 1992.

[ACD93] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[ACH+92] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, David L. Dill, and Howard Wong-Toi. Minimization of timed transition systems. In *Proceedings of the Third International Conference on Concurrency Theory (CONCUR '92)*, pages 340–354, London, UK, 1992. Springer-Verlag.

[AD89] André Arnold and Anne Dicky. An algebraic characterization of transition system equivalences. *Information and Computation*, 82(2):198–229, 1989.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[AFH96] Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.

[AFH99] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theoretical Computer Science*, 211(1-2):253–273, 1999.

[AH93] Rajeev Alur and Thomas A. Henzinger. Real-time logics: complexity and expressiveness. *Information and Computation*, 104:390–401, 1993.

[AH94] Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–203, 1994.

[AMP95] Eugene Asarin, Oded Maler, and Amir Pnueli. Symbolic controller synthesis for discrete and timed systems. In *Hybrid Systems II*, Lecture Notes in Computer Science, pages 1–20, London, UK, 1995. Springer-Verlag.

[AN01] André Arnold and Damian Niwiński. *Rudiments of μ-calculus*, volume 146 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 2001.

[AVW03] André Arnold, Aymeric Vincent, and Igor Walukiewicz. Games for synthesis of controllers with partial observation. *Theoretical Computer Science*, 303(1):7–34, 2003.

[AW07] André Arnold and Igor Walukiewicz. Nondeterministic controllers of nondeterministic processes. In *Logic and Automata*, Texts in Logic and Games, pages 29–52. Amsterdam University Press, 2007.

[B$\ddot{6}$2] J. R. Büchi. On a decision method in restricted second-order arithmetic. In *Proc. 1960 Int. Congr. for Logic, Methodology, and Philosophy of Science*, pages 1–1. Stanford Univ. Press, 1962.

[BBC06] Patricia Bouyer, Laura Bozzelli, and Fabrice Chevalier. Controller synthesis for MTL specifications. In *Proceedings of the 17th International Conference on Concurrency Theory (CONCUR'06)*, volume 4137 of *Lecture Notes in Computer Science*, pages 450–464. Springer-Verlag, Bonn, Germany, august 2006.

[BBF+01] Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Petit, Laure Petrucci, and Philippe Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer-Verlag, 2001.

[BC96] Girish Bhat and Rance Cleaveland. Efficient local model checking for fragments of the modal calculus. In *Proceedings of the 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, volume 1055 of *Lecture Notes in Computer Science*, pages 107–126. Springer-Verlag, 1996.

[BC05] Patricia Bouyer and Fabrice Chevalier. On conciseness of extensions of timed automata. *Journal of Automata, Languages and Combinatorics*, 10(4):393–405, 2005.

[BCL05] Patricia Bouyer, Franck Cassez, and François Laroussinie. Modal logics for timed control. In *Proceedings of the 16th International Conference on Concurrency Theory (CONCUR'05)*, volume 3653 of *Lecture Notes in Computer Science*, pages 81–94, San Francisco, CA, USA, august 2005. Springer.

[BDFP04] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2-3):291–345, august 2004.

[BDGP98] Béatrice Bérard, Volker Diekert, Paul Gastin, and Antoine Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2):145–182, November 1998.

[BDL04] Gerd Behrmann, Alexandre David, and Kim G. Larsen. A tutorial on UPPAAL. In *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM-RT '04)*, volume 3185 of *Lecture Notes in Computer Science*, pages 200–236. Springer–Verlag, September 2004.

[BDM+98] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. Kronos: A model-checking tool for real-time systems (tool-presentation for ftrtft '98). In *Proceedings of the 5th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'98)*, pages 298–302, London, UK, 1998. Springer-Verlag.

[BDMP03] Patricia Bouyer, Deepak D'Souza, P. Madhusudan, and Antoine Petit. Timed control with partial observability. In Warren A. Hunt, Jr and Fabio Somenzi, editors, *Proceedings of the 15th International Conference on Computer Aided Verification (CAV'03)*, volume 2725 of *Lecture Notes in Computer Science*, pages 180–192, Boulder, Colorado, USA, july 2003. Springer.

[Bek84] Hans Bekic. Definable operation in general algebras, and the theory of automata and flowcharts. In *Programming Languages and Their Definition - Hans Bekic*, pages 30–55, London, UK, 1984. Springer-Verlag.

[BK06] Samik Basu and Ratnesh Kumar. Quotient-based control synthesis for non-deterministic plants with μ-calculus specifications. In *Proceedings of the 45th Conference on Decision and Control*, 2006.

[BL05] Dietmar Berwanger and Giacomo Lenzi. The variable hierarchy of the μ-calculus is strict. In *Proceedings of the 22nd Symposium on Theoretical Aspects of Computer Science (STACS'05)*, volume 3404 of *LNCS*, pages 97–109. Springer-Verlag, 2005.

[BLL+96] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Uppaal—a tool suite for automatic verification of real-time systems. In *Proceedings of the DIMACS/SYCON workshop on Hybrid systems III : verification and control*, pages 232–243, Secaucus, NJ, USA, 1996. Springer-Verlag New York, Inc.

[BLR05] Patricia Bouyer, François Laroussinie, and Pierre-Alain Reynier. Diagonal constraints in timed automata: Forward analysis of timed systems. In Paul Pettersson and Wang Yi, editors, *Proceedings of the 3rd International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'05)*, volume 3829 of *Lecture Notes in Computer Science*, pages 112–126, Uppsala, Sweden, november 2005. Springer.

[Bou03] Patricia Bouyer. Untameable timed automata! In Helmut Alt and Michel Habib, editors, *Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS'03)*, volume 2607 of *Lecture Notes in Computer Science*, pages 620–631, Berlin, Germany, february 2003. Springer.

[Bra98] J. C. Bradfield. The modal μ-calculus alternation hierarchy is strict. *Theoretical Computer Science*, 195(2):133–153, 1998.

[BTY97] Ahmed Bouajjani, Stavros Tripakis, and Sergio Yovine. On-the-fly symbolic model checking for real-time systems. In *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS '97)*, pages 232–243. IEEE Computer Society, 1997.

[BY04] Johan Bengtsson and Wang Yi. *Timed Automata: Semantics, Algorithm and tools*, pages 87–124. Lecture Notes in Computer Science. Springer Berlin/Hidelberg, 2004.

[Cas87] Ilaria Castellani. Bisimulations and abstraction homomorphisms. *Journal of Computer and System Sciences*, 34(2/3):210–235, 1987.

[CCG00] Sérgio Campos, Edmund M. Clarke, and Orna Grumberg. Selective quantitative analysis and interval model checking: Verifying different facets of a system. *Formal Methods in System Design*, 17(2):163–192, 2000.

[CCG+02] Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In *Proceedings of the 14th International Conference on Computer Aided Verification(CAV '02)*, pages 359–364, London, UK, 2002. Springer-Verlag.

[CE82] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK, 1982. Springer-Verlag.

[Cer93] Karlis Cerans. Decidability of bisimulation equivalences for parallel timer processes. In *Proceedings of the 4th International Workshop on Computer Aided Verification (CAV '92)*, pages 302–315, London, UK, 1993. Springer-Verlag.

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, Cambridge, MA, USA, 1999.

[CPHP87] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice. Lustre: a declarative language for real-time programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of Programming Languages (POPL'87)*, pages 178–188, New York, NY, USA, 1987. ACM.

[Dil90] David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proceedings of the international workshop on Automatic verification methods for finite state systems*, pages 197–212, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[DM02] Deepak D'Souza and P. Madhusudan. Timed control synthesis for external specifications. In *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS '02)*, pages 571–582, London, UK, 2002. Springer-Verlag.

[EH83] E. Allen Emerson and Joseph Y. Halpern. "sometimes" and "not never" revisited: on branching versus linear time (preliminary report). In *Proceedings of the 10th ACM SIGACT-SIGPLAN symposium on Principles of programming languages (POPL '83:)*, pages 127–140, New York, NY, USA, 1983. ACM.

[EJ91] E. Allen Emerson and Charanjit S. Jutla. Tree automata, μ-calculus and determinacy. In *Proceedings of the 32nd annual symposium on Foundations of computer science (SFCS '91)*, pages 368–377, Washington, DC, USA, 1991. IEEE Computer Society.

[Eme85] E. Allen Emerson. Automata, tableaux and temporal logics (extended abstract). In *Proceedings of the Conference on Logic of Programs*, pages 79–88, London, UK, 1985. Springer-Verlag.

[Eme90] E. Allen Emerson. *Temporal and modal logic*, volume B: formal models and semantics, pages 995–1072. MIT Press, Cambridge, MA, USA, 1990.

[Eme97] E. Allen Emerson. Model checking and the mu-calculus. In *DIMACS Series in Discrete Mathematics*, pages 185–214. American Mathematical Society, 1997.

[EMSS91] E. Allen Emerson, Aloysius K. Mok, A. Prasad Sistla, and Jai Srinivasan. Quantitative temporal reasoning. In *Proceedings of the 2nd International Workshop on Computer Aided Verification (CAV '90)*, pages 136–145, London, UK, 1991. Springer-Verlag.

[Fle02] Emmanuel Fleury. *Les automates temporisés avec mises à jour*. Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, December 2002.

[Flo62] Robert W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.

[Fre05] Goran Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. In *Proceedings of the 8th International Workshop Hybrid Systems : Computation and Control (HSCC'05)*, pages 258–273, 2005.

[Fre08] Goran Frehse. Phaver: algorithmic verification of hybrid systems past hytech. *International Journal on Software Tools for Technology Transfer*, 10(3):263–279, 2008.

[Gen34] Gerhard Gentzen. Untersuchungen über das logische schließen i. *Mathematische Zeitschrift*, 39(1)(39):176–210, 1934.

[GH82] Yuri Gurevich and Leo Harrington. Trees, automata, and games. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing (STOC '82)*, pages 60–65, New York, NY, USA, 1982. ACM.

[GHKK05] Hermann Gruber, Markus Holzer, Astrid Kiehn, and Barbara König. On timed automata with discrete time - structural and language theoretical characterization. In *Developments in Language Theory (DLT '05)*, pages 272–283, 2005.

[GM96] Giuseppe De Giacomo and Fabio Massacci. Tableaux and algorithms for propositional dynamic logic with converse. In *Proceedings of the 13th International Conference on Automated Deduction (CADE-13)*, volume 1104 of *Lecture notes in artificial intelligence (LNAI)*, pages 613–628. Springer, 1996.

[Gor99] Rajeev Gore. *Tableau methods for modal and temporal logics*, pages 297–398. Springer, 1999.

[GV04] Alain Griffault and Aymeric Vincent. The mec 5 model-checker. In *Proceedings of the 16th International Conference on Computer Aided Verification(CAV'04)*, volume 3114 of *Lecture Notes in Computer Science*, pages 488–491. Springer, july 2004.

[GV08] Orna Grumberg and Helmut Veith, editors. *25 Years of Model Checking - History, Achievements, Perspectives*, volume 5000 of *Lecture Notes in Computer Science*. Springer, 2008.

[Hen96] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS,96)*, pages 278–292, Washington, DC, USA, 1996. IEEE Computer Society.

[HHWT97] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: A model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer*, 1(1–2):110–122, 1997.

[HLY91] Uno Holmer, Kim Larsen, and Wang Yi. Deciding properties of regular real timed processes. In *Proceedings of the 3th International Conference on Computer Aided Verification (CAV '91)*, volume 575 of *Lecture Notes in Computer Science*, pages 432–442. Springer-Verlag, 1991.

[HM80] Matthew Hennessy and Robin Milner. On observing nondeterminism and concurrency. In *Proceedings of the 7th International Colloquium on Automata, Languages and Programming (ICALP'80)*, pages 299–309, London, UK, 1980. Springer-Verlag.

[HMP92] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. Timed transition systems. In *Proceedings of the Real-Time: Theory in Practice, REX Workshop*, pages 226–251, London, UK, 1992. Springer-Verlag.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[Hol97] Gerard J. Holzmann. The model checker spin. *IEEE IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[HP85] David Harel and Amir Pnueli. *On the development of reactive systems*, pages 477–498. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[HR04] Yoram Hirshfeld and Alexander Rabinovich. Logics for real time: Decidability and complexity. *Fundamenta Informaticae*, 62(1):1–28, 2004.

[IPPA00] Anna Ingólfsdóttir, Mikkel Lykke Pedersen, Jan Poulsen, and Luca Aceto. Characteristic formulae for timed automata. *RAIRO, Theoretical Informatics and Applications*, 34(34):565–584, 2000.

[Jur00] Marcin Jurdzinski. Small progress measures for solving parity games. In *Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science(STACS '00)*, pages 290–301, London, UK, 2000. Springer-Verlag.

[JW95] David Janin and Igor Walukiewicz. Automata for the modal μ-calculus and related results. In *Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Science (MFCS '95)*, pages 552–562, London, UK, 1995. Springer-Verlag.

[KL96] Inhye Kang and Insup Lee. An efficient state space generation for analysis of real-time systems. *SIGSOFT Software Engineering Notes*, 21(3):4–13, 1996.

[Koy90] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time System*, 2(4):255–299, 1990.

[Koz82] Dexter Kozen. Results on the propositional μ-calculus. In *Proceedings of the 9th International Colloquium on Automata, Languages and Programming (ICALP'82)*, pages 348–359, 1982.

[KVW00] Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

[Lö9] Christof Löding. Optimal bounds for transformations of ω-automata. In *Proceedings of the 19th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'99)*, pages 97–109, London, UK, 1999. Springer-Verlag.

[Lar05] François Laroussinie. *Model checking temporisé — Algorithmes efficaces et complexité*. Mémoire d'habilitation, Université Paris 7, Paris, France, december 2005.

[LL98] François Laroussinie and Kim G. Larsen. CMC: A tool for compositional model-checking of real-time systems. In *Proceedings of IFIP Joint Int. Conf. Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE-PSTV'98), Paris, France, Nov. 1998*, pages 439–456. Kluwer Academic, 1998.

[LLW95] François Laroussinie, Kim G. Larsen, and Carsten Weise. From timed automata to logic - and back. In *Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Science (MFCS '95)*, pages 529–539, London, UK, 1995. Springer-Verlag.

[LMS04] François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. Model checking timed automata with one or two clocks. In *Proceedings of the 15th International Conference on Concurrency Theory (CONCUR'04)*, volume 3170 of *Lecture Notes in Computer Science (LNCS)*, pages 387–401, London, UK, august 2004. Springer.

[LP85] Orna Lichtenstein and Amir Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the 12th ACM SIGACT-SIGPLAN symposium on Principles of programming languages (POPL '85)*, pages 97–107, New York, NY, USA, 1985. ACM.

[LPY97] Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1:134–152, 1997.

[LPZ85] Orna Lichtenstein, Amir Pnueli, and Lenore D. Zuck. The glory of the past. In *Proceedings of the Conference on Logic of Programs*, pages 196–218, London, UK, 1985. Springer-Verlag.

[LS01] Martin Lange and Colin Stirling. Focus games for satisfiability and completeness of temporal logic. In *Proceedings of the 16th Annual IEEE Symposium on Logic*

in Computer Science(LICS '01), page 357, Washington, DC, USA, 2001. IEEE Computer Society.

[LY97] Kim G. Larsen and Wang Yi. Time abstracted bisimulation: Implicit specification and decidability. *Information and Computation*, 134:75–103, 1997.

[Mer01] Stephan Merz. Model checking: A tutorial overview. In F. Cassez et al., editor, *Modeling and Verification of Parallel Processes*, volume 2067 of *Lecture Notes in Computer Science*, pages 3–38. Springer-Verlag, Berlin, 2001.

[Mos85] A. W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In *The 5th Symposium on Computation Theory*, volume 208 of *Lecture Notes in Computer Science*, pages 157–168. Springer-Verlag, 1985.

[MS87] David E. Muller and Paul E. Schupp. Alternating automata on infinite trees. *Theor. Comput. Sci.*, 54:267–276, 1987.

[MS95] 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. *Theoretical Computer Science*, 141(1-2):69–107, 1995.

[Niw88] Damian Niwinski. Fixed points vs. infinite generation. In *Proceedings of the Third Annual Symposium on Logic in Computer Science (LICS'88)*, pages 402–409, 1988.

[NW96] Damian Niwiński and Igor Walukiewicz. Games for the μ-calculus. *Theoretical Computer Science*, 163(1-2):99–116, 1996.

[Ong02] C.-H. Luke Ong. Model checking algol-like languages using game semantics. In *Proceedings of the 22nd Conference Kanpur on Foundations of Software Technology and Theoretical Computer Science (FST TCS '02)*, pages 33–36, London, UK, 2002. Springer-Verlag.

[OW05] Joël Ouaknine and James Worrell. On the decidability of metric temporal logic. In *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science (LICS '05)*, pages 188–197, Washington, DC, USA, 2005. IEEE Computer Society.

[OW06a] Joël Ouaknine and James Worrell. On metric temporal logic and faulty turing machines. In *Proceedings of the 9th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'06)*, pages 217–230, 2006.

[OW06b] Joël Ouaknine and James Worrell. Safety metric temporal logic is fully decidable. In *Proceedings of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2006.

[Par81] David Park. Concurrency and automata on infinite sequences. In *Proceedings of the 5th GI-Conference on Theoretical Computer Science*, pages 167–183, London, UK, 1981. Springer-Verlag.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual IEEE Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57, 1977.

[PR05] Sophie Pinchinat and Stéphane Riedweg. A decidable class of problems for control under partial observation. *Information Processing Letter*, 95(4):454–460, 2005.

[PZ93] Amir Pnueli and Lenore D. Zuck. Probabilistic verification. *Information and Computation*, 103(1):1–29, 1993.

[Rab69] Michael O. Rabin. Decidability of second order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[RS99] 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.

[RW89] Peter J. Ramadge and W. Murray Wohnam. The control of discrete event systems. In *Proceedings of the IEEE Computer Society*, volume 77, pages 81–98, 1989.

[Sav01] Alexandru Tiberiu Sava. *Sur la synthèse de la commande des systèmes à événements discrets temporisés*. PhD thesis, Institut National Polytechnique de Grenoble - INPG, 11 2001.

[Sch03] Philippe Schnoebelen. The complexity of temporal logic model checking. In *Selected Papers from the 4th Workshop on Advances in Modal Logics (AiML'02)*, pages 393–436, Toulouse, France, 2003. King's College Publication.

[SE89] Robert S. Streett and E. Allen Emerson. An automata theoretic decision procedure for the propositional μ-calculus. *Information and Computation*, 81(3):249–264, 1989.

[SI94] Bernhard Steffen and Anna Ingólfsdóttir. Characteristic formulae for processes with divergence. *Information and Computation*, 110(1):149–163, 1994.

[Sif01] Joseph Sifakis. Modeling real-time systems-challenges and work directions. In *Proceedings of the First International Workshop on Embedded Software(EMSOFT '01)*, pages 373–389, London, UK, 2001. Springer-Verlag.

[Sor01] Maria Sorea. Tempo: A model-checker for event-recording automata. In *Proceedings of the 2nd Workshop on Real-Time Tools (RT-TOOLS'01)*, Aalborg, Denmark, August 2001.

[Sor02] Maria Sorea. A decidable fixpoint logic for time-outs. In *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR '02)*, pages 255–271, London, UK, 2002. Springer-Verlag.

[SS94] Oleg V. Sokolsky and Scott A. Smolka. Incremental model checking in the modal μ-calculus. In *Proceedings of the 6th International Conference on Computer Aided Verification (CAV '94)*, pages 351–363. Springer-Verlag, 1994.

[SS95] Oleg Sokolsky and Scott A. Smolka. Local model checking for real-time systems (extended abstract). In *Proceedings of the 7th International Conference on Computer Aided Verification (CAV'95)*, pages 211–224, London, UK, 1995. Springer-Verlag.

[Sti96] Colin Stirling. Modal and temporal logics for processes. In *Proceedings of the VIII Banff Higher order workshop conference on Logics for concurrency : structure versus automata*, pages 149–237, Secaucus, NJ, USA, 1996. Springer-Verlag New York, Inc.

[Tar55] Alfred Tarski. A lattice theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[Tho90] Wolfgang Thomas. Automata on infinite objects. In *Handbook of theoretical computer science*, volume B: formal models and semantics, pages 133–191. MIT Press, Cambridge, MA, USA, 1990.

[Tho97] Wolfgang Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3. Springer-Verlag, 1997.

[TXJS92] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic Model Checking for Real-Time Systems. In *Proceedings of the 7th. Symposium of Logics in Computer Science(LICS'92)*, pages 394–406, Santa-Cruz, California, 1992. IEEE Computer Scienty Press.

[TY01] Stavros Tripakis and Sergio Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Description*, 18(1):25–68, 2001.

[Var96] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Proceedings of the VIII Banff Higher order workshop conference on Logics for concurrency : structure versus automata*, pages 238–266, Secaucus, NJ, USA, 1996. Springer-Verlag New York, Inc.

[Var07] Moshe Y. Vardi. Automata-theoretic model checking revisited. In *Proceedings of the 8th International Conference on Verification Model Checking and Abstract Interpretation (VMCAI'07)*, volume 4349 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 137–150. Springer-Verlag, 2007.

[vG97] R. J. van Glabbeek. Notes on the methodology of CCS and CSP. In *Proceedings from the international workshop on Algebra of communicating processes (ACP '95)*, pages 329–349, Amsterdam, The Netherlands, The Netherlands, 1997. Elsevier Science Publishers B. V.

[VJ00] Jens Vöge and Marcin Jurdziński. A discrete strategy improvement algorithm for solving parity games. In *Proceedings of the 12th International Conference on Computer Aided Verification(CAV '00)*, pages 202–215, London, UK, 2000. Springer-Verlag.

[Wal95] Igor Walukiewicz. Notes on the propositional μ-calculus: Completeness and related results. Technical Report NS-95-1, Aarhus University, Basic Research In Computer Sciences, 1995.

[Wal01] Igor Walukiewicz. Automata and logic,notes from eff summer school'01, 2001.

[Wil01] Thomas Wilke. Alternating tree automata, parity games, and modal μ-calculus. *Bulletin of the Belgian Mathematical Society*, 8(2), May 2001.

[Yi90] Wang Yi. Real-time behaviour of asynchronous agents. In *CONCUR '90 Proceedings on Theories of concurrency : unification and extension*, pages 502–520, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[YL97] Mihalis Yannakakis and David Lee. An efficient algorithm for minimizing real-time transition systems. *Form. Methods Syst. Des.*, 11(2):113–136, 1997.

[Yov98] Sergio Yovine. Model checking timed automata. In *Lectures on Embedded Systems, European Educational Forum, School on Embedded Systems*, pages 114–152, London, UK, 1998. Springer-Verlag.

[Zie98] Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998.

Index