CONTROLLER SYNTHESIS FOR REACTIVE SYSTEMS IN DISTRIBUTED, REAL-TIME AND HYBRID SETTINGS

YANG SHAOFA
(M.Sc., NUS)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2006

Acknowledgements

I am deeply grateful to Professor P. S. Thiagarajan, my supervisor, for his excellent guidance and valuable advices. Through research in this thesis, I have learnt a lot from him.

Parts of the results in this thesis were jointly obtained with P. Madhusudan, P. S. Thiagarajan and Wang Yi. I feel privileged for having been given the chance to work with them. And I am grateful to them for their prolific ideas.

I thank Associate Professors Chin Wei Ngan and Dong Jin Song for their valuable comments on my Qualifying Exam and Thesis Proposal reports.

I thank my institution, School of Computing, National University of Singapore, for supporting my PhD candidature with a teaching assistantship.

Contents

Summary

An *open* system is one which repeatedly interacts with an environment, and whose behaviour crucially depends on this interaction. The subject of controller synthesis deals with automatic construction of controllers for open systems. In sequential settings, the controller synthesis problem is: Given a *plant*, which describes the possible interactions between the system and the environment, and a *specification*, that dictates the desired behaviour, determine whether there exists a *controller* such that the controlled behaviour of the plant satisfies the specification. The goal of this thesis is to investigate controller synthesis problems in distributed, real-time and hybrid settings.

Distributed Setting

The distributed controller synthesis problem is: Given a *distributed plant* and a specification, determine whether there exists a *distributed controller* such that the overall controlled behaviour of the distributed plant satisfies the specification. A distributed plant consists of a family of open sequential processes communicating with each other, where each process interacts with its local environment. A distributed controller consists of a family of *local strategies*, one for each process. The local strategy for process p recommends moves for p, based on the knowledge of actions executed by p as well as actions executed by other processes that p comes to know via communication.

Distributed controller synthesis problems are undecidable in general settings [62], but are decidable in various restricted settings [26, 39, 48, 49, 51, 62]. We study a setting where the *communication pattern* of the distributed plant is restricted. We identify the model of *connectedly communicating processes (CCP)*. A CCP consists of a network of sequential processes which communicate via synchronizing on common actions. And there exists a bound k such that, for every process p, q, if p executes k steps without hearing from q, directly or indirectly, then p will never hear from q again, directly or indirectly.

The non-interleaved branching time behaviour of a CCP is captured by its event structure unfolding. We prove that the monadic second order (MSO) theory of the event structure unfolding of every CCP is decidable. Using this strong logical result, we establish three results on the distributed controller synthesis problem for distributed plants based on CCPs. Firstly, we show that the problem is decidable for *robust linear time specifications* that do not discriminate different interleavings of the same partially ordered execution. Secondly, we prove that the problem is also decidable for *branching time specifications* given as formulae in the MSO logic of the event structure unfolding of the given CCP plant. Lastly, for both the first and second results, we further establish that, if there exists a distributed controller, then a *finite state* one can be effectively synthesized in the form of a CCP.

On the negative side, we show that the distributed controller synthesis problem for CCP plants is undecidable for linear time specifications that are allowed to be *non-robust*. We also study the *strict* distributed controller synthesis problem where one seeks a family of *strictly local* strategies, one for each process. A strictly local strategy for process p must recommend moves for p based *only* on the knowledge of actions executed by p. We prove that the strict distributed controller synthesis for CCP plants is undecidable for linear time specifications, even if they are robust.

Real-Time Setting

There have been a number of studies that extend results on sequential controller synthesis to timed settings [9, 13, 19, 53]. We however are interested in real-time systems with *tasks*. The correctness of many real-time systems depends not only on the timely occurrence of events, but also on the proper handling of computation tasks triggered by events. With a fixed computing resource and a fixed scheduling policy, a real-time system may *not* be schedulable in the sense that not every task instance can be completed before its deadline. We address this problem systematically by synthesizing an *admission controller*. Upon every newly arrived task instance, the admission controller either accepts it and puts it into the ready queue through the scheduling policy, or rejects (discards) it. We demand that every accepted task instance must be completed before its deadline, and moreover, the task acceptance pattern must satisfy a quality-of-service (QoS) specification.

We consider the uniprocessor setting with the preemptive EDF (earliest-deadline-first) scheduling policy. We adopt the generic approach by [24] of modelling the task arrival pattern of a real-time system using a timed automaton. We prove that the admission controller synthesis problem is decidable for QoS specifications given as *linear time temporal logic* (LTL) formulae, and more generally for QoS specifications given as *quantified propositional linear time temporal logic* (QPLTL) formulae. In both cases, we further show that if an admission controller exists, then we can effectively synthesize one in the form of a (finite) timed automaton.

Using LTL formulae, we can specify that instances of task τ must always be accepted. We can assert liveness properties. For example, instances of task τ must be accepted infinitely often. We can also dictate fairness properties. For example, if instances of task τ are accepted infinitely often, then so are instances of task τ'. For a *fixed* integer n, we can demand that among every n consecutive instances of task τ, at least $.7n$ must be accepted. Using QPLTL formulae, we can require that, for a *fixed* integer n, every n-th instance of task τ must be accepted, while other instances of task τ may or may not be accepted. However, it seems that, in LTL or QPLTL, we can not express properties such as that the *limit* of the acceptance ratio of instances of task τ is at least .7.

Hybrid Setting

A hybrid automaton models a digital control system interacting with a continuous environment. Basically, a hybrid automaton consists of finitely many control states and a transition relation between them. The continuous environment is represented by finitely many real-valued variables. At each control state, the variables evolve according to some differential equation. A transition is associated with a guard in terms of the variables and can be taken only when the guard is true. The most basic question about a hybrid automaton is the reachability problem, which is to determine whether a designated control state can ever be reached.

The continuous time semantics for hybrid automata allows a transition to be taken at any real-valued time. As a result, the reachability problem is undecidable in general [32], except for variants of hybrid automata which have the feature that values of variables are *reset* when a transition is taken [6, 32, 42, 43]. We believe that this resetting feature severely limits the kind of practical control systems that can be modelled. On the other hand, the discrete time semantics demands that a transition can occur only at integer time instants. Under the discrete time semantics, the reachability problem is decidable for subclasses of hybrid automata whose key restriction is that the rates of variables are constant ($dx/dt = c$) [3, 4, 30].

We propose the class of *restricted differential hybrid automata (RDA)*. Its key feature is that the rates of variables can either be constant or exponential ($dx/dt = c \cdot x$). We adopt the discrete time semantics. However, as in [3, 4], we allow the *sensing* of values of variables and *updating* of rates of variables to occur with bounded delays. We prove that the language of control state sequences of an RDA is regular. This implies that the reachability for RDAs is decidable. Using the regularity result, we show that if there is *no* sensing delay, then the controller synthesis problem for RDAs is decidable for linear time specifications given as LTL formulae. Further, we show that if a controller exists, then we can effectively synthesize one in the form of a (finite) RDA. The obstacle of tackling controller synthesis for RDAs is that the controller has *incomplete* information about the values of variables due to the presence of sensing delays.

CHAPTER 1

Introduction

In this introductory chapter, we first give the motivation of controller synthesis in section 1.1. Subsequently, in section 1.2, we review the historical background and the literature on sequential controller synthesis. In section 1.3, we give an overview of our contributions on controller synthesis in distributed, real-time and hybrid settings. In the last section, we outline the organization of subsequent chapters.

1.1. Controller Synthesis

Computing devices are widely used in many safety-critical applications such as aircrafts, nuclear reactors, and so on. The correct functioning of these computing devices is of paramount importance. Many of these devices are *reactive* in the sense that they repeatedly interact with physical environments and their behaviours crucially depend on these interactions. For example, a car brake controller constantly monitors the car's speed and other parameters and activates a brake or other actions whenever necessary. The construction of reactive systems has been a difficult problem, since one needs to design them with infinite behaviours in mind. What can we do if a constructed reactive system does not satisfy some property? One may ask an ambitious question: Given a constructed reactive system, and a *specification* of correct behaviour, can we automatically synthesize a *controller* that restricts the system so that the controlled behaviour satisfies the specification, no matter what the the environment does? This is the *controller synthesis* problem. The given reactive system is typically called a *plant* in this context.

Besides the computer science community, the control theory community has also studied the controller synthesis problem but call it *supervisory control of discrete event systems*. These two communities have different viewpoints on the problem, as we will describe in detail in the next section. In this thesis, we adopt the viewpoint of the computer science community.

In what follows, we describe informally the controller synthesis problem in sequential settings and the associated concepts. A mathematically precise formulation will be given in section 2.3.

In the sequential setting, a *plant* can be represented as a finite bipartite graph whose state (vertex) set is partitioned into environment and system states. For each environment state s, its successor states represent the possible moves that the environment may make at s. For each system state s, its successor states represent the possible choices of moves available to the system.

A (linear time) specification is basically an ω-regular language over the action alphabet of the plant. Such a specification may be presented, say, as a non-deterministic Büchi automaton.

The notion of a controller is based on a *strategy*. At each stage when the plant is in a system state, a strategy shall advise the system what moves to take next. The recommendation of the strategy is based on the current history of actions executed by the system and the environment. The strategy must recommend the system only moves that are possible as indicated by the plant description. If we reach a stage where it is the environment's turn to make a move, then the strategy must allow *all* possible moves of the environment. We also demand the strategy to be *non-blocking*. More precisely, whenever the system reaches a stage by following recommendations of the strategy, there will always be moves that the system can make and that are also recommended by the strategy. We note that this notion is different from and in fact weaker than that in supervisory control of discrete event systems [67].

An infinite play is an infinite sequence of actions of the system and environment that are possible from the plant description. An infinite play σ is *according to* a strategy f iff the moves made by the system in σ are always inside the corresponding recommendations by f.

We say a strategy f is winning iff f is non-blocking and every infinite play according to f falls within the specification. By a *controller*, we shall mean a winning strategy.

The controller synthesis problem can now be more precisely stated: Given a plant and a specification, does there exist a controller? This problem has been answered in the affirmative in many sequential settings. The foundation for these solutions is the decidability of the monadic second order (MSO) theory of n-successors interpreted over tree unfoldings of finite transition systems. The tree unfolding of a finite transition system represents its branching time behaviour. This logical result follows from Rabin's famous theorem [63], which states that the MSO theory of 2-successors is decidable.

Loosely speaking, in the sequential setting where the plant is a finite transition system and the specification is an ω-regular language, we can effectively construct a sentence φ in the MSO logic of n-successors interpreted over the tree unfolding of the plant, such that φ is true iff there exists a controller. Hence by testing the truth of φ, we can determine whether there exists a controller. Further, in case φ is true, the decision procedure for testing the truth of φ also yields a *regular* witness, which can then be viewed as a finite state controller.

We emphasize that even for controller synthesis with linear time specifications, one has to study the branching time behaviour of the plant in order to determine the existence of a winning strategy. This is due to that, at all environment states, the strategy must allow *all* moves that could possibly be made by the environment.

Technically, the solutions for controller synthesis problems are quite intricate and usually employ sophisticated machineries from automata theory. Moreover, the worst case complexities of these solutions are so high that they still do not seem feasible to be implemented practically. The search of practically feasible algorithms for controller synthesis has been a real challenge for the research community and is a long term goal. However, the realization of this goal is not hopeless, since one would reasonably expect that the theoretical worst cases for these decision procedures rarely occur in practice.

Our goals in this thesis are to explore controller synthesis problems in distributed, real-time and hybrid settings. We are interested mainly in theoretical aspects.

1.2. Related Work on Controller Synthesis

Here we review related work on sequential controller synthesis from both the computer science and control theory communities.

In computer science, the controller synthesis problem is closely related to the *realizability problem*. Loosely speaking, the realizability problem is: Given a specification over an alphabet of environment and system actions, does there exist a reactive program whose behaviour satisfies the specification? In other words, the aim of the realizability problem is to synthesize a reactive program from a specification. On the other hand, controller synthesis is concerned with restricting an already constructed reactive system, that is, the plant, so that a specification is met. Technically, the realizability problem and the controller synthesis problem can often be tackled using similar tools. Often, the realizability problem can be viewed as a special case of the controller synthesis problem if the formulation of a "universal" plant, that allows all possible interactions of the system and the environment, is available. For example, for the sequential setting described in the previous section, a universal plant can be represented as a *complete* bipartite graph, where the successor states of an environment state are all the system states and conversely, the successor states of a system state are all the environment states.

In this thesis, we study only the controller synthesis problem. We believe that it is more widely applicable than the realizability problem. Note that in order to synthesize a full reactive system from a specification, the specification has to describe *all* aspects of this reactive system. This is not practical in most cases. On the other hand, in the controller synthesis problem, the objective is to restrict an already constructed reactive system so that *some* specific property is satisfied.

The realizability problem was first posed by Church [16] in 1963 in the context of synthesizing switching circuits against specifications stated in restricted second-order arithmetic. This was solved positively by Büchi and Landweber [14], but later dealt with more elegantly by Rabin [64] (see also [74]) using tree automata.

In the eighties, several works [22, 54, 55] studied the automatic synthesis of finite state programs against temporal logic specifications. However, they consider *closed systems*. In other words, the program that one seeks against a temporal logic specification does not interact with an environment and hence everything about the program can be controlled. In essence, these papers solve the satisfiability problem for temporal logic formulae by determining whether there exist finite state programs that are witnesses to the given temporal logic formulae. Therefore, the results of [22, 54, 55] are not applicable to the realizability or the controller synthesis problem, where the environment is a crucial component.

The realizability problem was taken up later by [61], which investigated the complexity of synthesizing finite programs from LTL (linear time temporal logic) formulae using automata-theoretic techniques. Meanwhile, [56] studied infinite games played over finite graphs. The results of [56] are technically relevant to both the realizability problem and the controller synthesis problem.

The work [40] investigates the realizability problem for linear time specifications but considers the issue of partial observation. Namely, a strategy sees only executed actions that belong to a prescribed set of observable action alphabet.

The work [38] considers the controller synthesis problem for branching time specifications given as CTL (computation tree logic), CTL* ([21]) formulae. A strategy is winning iff the computation tree generated from the controlled plant satisfies the given CTL or CTL* formula.

The work [48] studies controller synthesis for branching time specifications that are given as transition systems. A strategy is then said to be winning iff there is a behaviour-preserving simulation from the controlled branching time behaviour of the plant to the tree unfolding of the specification. The results of [48] were extended to bisimulations in [50].

In the control theory community, supervisory control of discrete event systems (DESs) is initiated by [65, 66]. A DES operates in accordance with abrupt occurrences at possibly unknown and irregular intervals, of physically events. Events in a DES are classified as controllable (which can be disabled) and uncontrollable (which can not be disabled). Hence a DES can be viewed as an open system. A supervisory controller is a function which disables certain controllable events at each stage, based upon the history of event occurrences. The *supervisory control problem* is to seek a controller such that no matter how the environment behaves, the controlled behaviour of the DES satisfies the specification. For a survey on supervisory control of DESs, we refer to [67].

The literature on supervisory control of DESs are mostly concerned with specifications that dictate *finite* behaviours. Also the specification is usually stated in terms of the plant itself. For example, certain bad state should be avoided or certain marked state should be reached and so on. The focus of the control theory community is on simple subclasses of supervisory control problems for which there exist tractable synthesis schemes. The complications in the settings where supervisory control was investigated mainly come from *partial observation*, where the controller has limited power of observing the plant; *least restrictive controllers* that pose least restriction on the plant; and *decentralized control*, where the plant is monolithic but one seeks a collection of controllers, each controlling a subset of actions; and so on.

In contrast, the computer science community mainly deals with specifications that talk about infinite behaviours. And often the specification is independent of the plant. The computer science community concentrates on investigating decidability and undecidability results.

1.3. Contributions

The goals of this thesis are to investigate controller synthesis problems in distributed, real-time and hybrid settings.

Distributed Setting

Distributed controller synthesis was initiated in [62] where a distributed plant is represented as an *architecture* consisting of a set of local sites connected through fixed communication channels. And each local site may communicate with its local environment also through fixed channels. To be precise, the work [62] studies the distributed realizability problem. This problem is: given a specification and an architecture, is there a family of programs, one for each local site, such that the collective behaviour satisfies the specification. Technically, the distributed realizability problem is closely related to the distributed controller synthesis problem, in the sense they can be often be solved using similar tools. It was shown in [62] that for linear time specifications, the distributed realizability problem is undecidable even for the simple architecture that consists of just two sites that do not have any communication channels between them. Since then, decidability results in distributed realizability and distributed controller synthesis for various subclasses of architectures have been obtained in [39, 49, 62].

Another line of work in distributed controller synthesis assumes a distributed plant to be given as a network of sequential processes of communicating with each other by synchronizing on common actions. The problem is then to find a distributed controller such that the collective controlled behaviour of the distributed plant meets the specification. A distributed controller consists of a family of local strategies, one for each process. The local strategy for p should recommend moves for p based on knowledge about actions of p as well as knowledge on actions executed by other processes that p comes to know via synchronizations, directly or indirectly.

In this line of work that processes communicate via synchronizations on common actions, one obtains decidability results by imposing restrictions on local strategies [51] and also by restricting the trace alphabet associated with the distributed plant [26]. In fact, the work [26] shows decidability results only for specifications that concern *finite* behaviours. On the other hand, since we study controller synthesis for reactive systems, we are interested only in specifications that talk about *infinite* behaviours.

In this thesis, we are interested in distributed controller synthesis where the distributed plant consists of processes communicating via synchronizations on common actions. We believe that this framework is more widely applicable for modelling practical distributed protocols, than the framework of an architecture. The reason is that in many distributed protocols, whether a process would communicate with another process and what the content of this communication would be, depend crucially on the current local state of the process. The architecture framework is not flexible because it demands that a local site (process) keeps reading to and writing from fixed channels at each state.

We shall model distributed plants based on asynchronous transition systems. We place restrictions on the *communication patterns* of distributed plants and study its consequence on the decidability of the distributed controller synthesis problem.

We identify the subclass of *connectedly communicating* asynchronous transition systems. We say an asynchronous transition system is connectedly communicating, iff there exists a bound k such that for every process p, q, if process p executes k steps without hearing from q, directly or indirectly, then it will never hear from q again, directly or indirectly. By *connectedly communicating processes (CCPs)*, we refer to the subclass of connected communicating asynchronous transition systems. CCPs can model naturally distributed protocols where processes communicate frequently with each other so that they maintain bounded loss of status on each other. Further, if the loss of process p on the status of q exceeds the given bound, then p will never obtain any further information about q. This kind of phenomenon often occurs in distributed protocols where if one process tries to establish links with another process, then it would give up after at most n attempts for some fixed integer n. For illustrative purpose, we shall give a natural example of connectedly communicating processes in section 3.5, which models two processes exchanging data through two buffers.

As noted in section 1.1, the foundation for solving many sequential controller synthesis problems is the logical result that the MSO theory of the tree unfolding of a sequential system is decidable. Note that the tree unfolding of a sequential system represents its branching time behaviour. The non-interleaved branching time behaviour of a CCP is given by its event structure unfolding [18]. One can define naturally an MSO logic over event structures. To provide the foundation for distributed controller synthesis associated with CCPs, we prove the logical result that the MSO theory of the event structure unfolding of every CCP is decidable. Using this strong logical result, we then establish decidability results of distributed controller synthesis problems associated with CCP plants for both robust linear time specifications and branching time specifications. We emphasize that this logical result is also of independent interest for model checking of distributed protocols that can be modelled as CCPs.

A linear time specification is an ω-regular language. A distributed controller is said to satisfy a linear time specification L iff every infinite run of the controlled plant is in L. We say the linear time specification L is *robust* iff it does not discriminate two different linearizations of the same partially ordered execution. Namely, if an infinite run σ is in L, and the infinite run σ' is in fact arising from the same partially ordered execution as σ, then σ' must also be in L. We show that: Given a CCP distributed plant and a *robust* linear time specification, one can effectively determine whether there exists a distributed controller. Further, if such a distributed controller exists, then a *finite state* one can be effectively synthesized in the form of a CCP.

A branching time specification for a CCP distributed plant is a formula in the MSO logic of the event structure unfolding of the CCP plant. A distributed controller is said to satisfy such a branching time specification φ, iff φ is true in the "sub-event structure" resulting from the overall controlled behaviour of distributed plant. We show that: Given a CCP distributed plant and a branching time specification, one can effectively determine whether there exists a distributed controller. Further, if such a distributed controller exists, then a *finite state* one can be effectively synthesized in the form of a CCP.

On the negative side, we show that the distributed controller synthesis associated with CCP distributed plants is undecidable for linear time specifications that are allowed to be non-robust.

We also study the *strict distributed controller synthesis problem* where one seeks a strict distributed controller. A strict distributed controller consists of a family of *strictly local strategies*, one for each process. A strictly local strategy for p should recommend moves for p, based on only the history of actions executed by p. We show that the strict distributed controller synthesis with CCP distributed plants is undecidable for linear time specifications, even if they are robust.

Real-Time Setting

We next investigate controller synthesis in real-time settings. There have been a number of studies that extend results on sequential controller synthesis to timed settings [9, 13, 19, 53]. We however are interested in real-time systems with tasks. We emphasize that the correctness of many real-time systems depends not only on the timely occurrence of events, but also on the proper handling of computation tasks triggered by events.

Our aim is to study the problem of synthesizing *admission controllers* for real-time systems with tasks. In many real-time computing environments, there are some tasks that are time-critical and others that are not. To ensure that every critical task is completed before its deadline, it may be necessary to deny entry into the ready queue for some non-critical tasks. We address this problem in the framework of controller synthesis. The environment's moves are the releases of task instances. Upon each newly released task instance, there are two choices available to the system, one is to accept it and hence putting it into the ready queue, and the other is to reject (discard) it. The goal is to come up with an *admission controller* such that no accepted task instance misses its deadline. And the task acceptance patterns generated by the admission controller satisfies a quality-of-service (QoS) specification.

We follow the approach in [24] of modelling the task arrival pattern in a real-time environment as a timed automaton ([7]) extended with *tasks*. Each task is associated a computation time and a relative deadline. We assume the uniprocessor setting with the preemptive EDF (earliest-deadline-first) scheduling policy.

Since we are dealing with reactive real-time systems, we consider QoS specifications that are given as LTL formulae, and more generally, quantified propositional LTL (QPLTL) formulae [21].

The admission controller synthesis problem can be more precisely stated as: Given a *task plant* based on timed automata with tasks and a QoS specification in LTL or QPLTL, does there exist an admission controller? We show that this problem is decidable for QoS specifications in LTL and in QPLTL. In both cases, we show further that if such an admission controller exists, then we can effectively synthesize one in the form of a (finite) timed automaton.

Using LTL formulae, we can specify that a task τ is *hard* by asserting that every instance of τ must be accepted. We can also specify *qualitative* QoS requirements that will typically assert liveness properties and fairness properties. For instance, we can say, along every infinite run, instances of task τ must be accepted infinitely often, if they are released infinitely often. One can also say that, if instances of task τ are accepted infinitely often, then instances of task τ' must also be accepted infinitely often, assuming that instances of both τ, τ' are released infinitely often.

More interestingly, one can also express in LTL *quantitative* QoS requirements that has a "boundedness" flavour. For instance, for a *fixed* integer n, we can assert in LTL that among every consecutive n arrivals of instances of task τ, at least $0.7n$ of them must be accepted.

In QPLTL, we can also express QoS properties like, for a fixed n, every n-th instance of τ must be accepted, while other instances of τ may or may not be accepted. This property is *not* expressible in LTL [80].

However, we do not know how to use LTL or QPLTL to capture quantitative QoS requirements that concern the limit average behaviour of task acceptance patterns. For example, such a QoS property may demand that the limit of the average acceptance ratio of instances of task τ is at least 0.7. We believe that tools from quantitative games [20, 82] would provide good starting points for handling such QoS properties.

Hybrid Setting

A hybrid automaton models a digital control system interacting with a continuous environment. The environment is captured by finitely many real-valued variables. The digital system measures the values of these variables through sensors and updates the rates of evolution of these variables via actuators. Basically, a hybrid automaton is a finite transition system, whose states are typically called control states, augmented with finitely many real-valued variables. At each control state, the variables evolve according to some differential equation. The variables would usually be governed by different equations in different control states. A transition is associated with a guard in terms of the variables and can be taken only when the guard is true. The most basic question about a hybrid automaton is the reachability problem, which is to determine whether a designated control state can ever be reached.

In the continuous time semantics, a transition may be taken at any real-valued time provided its associated guard is true. This endows hybrid automata with very rich behaviour, and consequently, the reachability problem is undecidable even for simple subclasses of hybrid automata where each variable evolves at constant rates [32] ($dx/dt = c$). Decidability results on the reachability problem are obtained in [6, 32, 42, 43] for the variant of hybrid automata which have the feature that values of continuous variables are reset during mode switches. We believe that the resetting feature severely limits the kind of practical control systems that can be modelled, since the essential feature of control systems is that one can only affect the values of variables by changing their evolution rates. In [35], the reachability problem is shown to be decidable for a subclass of hybrid automata where the rates of variables are constant and with a strong restriction on the structure of the transition relation.

On the other hand, [30] proposes the discrete time semantics which demand that transitions can only be taken at integer-valued time instants. With the discrete time semantics, [30] shows that the reachability problem is decidable for the class of hybrid automata where the rate of each variable could be any constant from a given interval, and the values of variables are within a prescribed range.

With the discrete time semantics, [3, 4] show further that the control state sequence language is regular for classes of hybrid automata with two key features. One is that variables evolve at constant rates. The other is that both sensing of values of variables and updating of rates of variables can take place within bounded delays from the integer time points.

We propose a class of hybrid automata, which we call *restricted differential hybrid automata (RDA)*. Its key feature is that variables can evolve at either constant rates, or exponential rates ($dx/dt = c \cdot x$). As in [3, 4], we adopt the discrete time semantics, but allow bounded delay in both sensing of values of variables and updating of rates of variables. We prove that the control state sequence language of an RDA is regular. This regularity result provides the foundation for studying controller synthesis problems with RDAs, though it is also of independent interest for model checking of RDAs. In [2], it is shown that the control state sequence languages for a variant of RDAs are regular. However, [2] does not study the controller synthesis problem.

We view an RDA naturally as a plant, which describes the possible interactions of the control system and the continuous environment. At each discrete time instant, a *strategy* for the plant should advise the system whether to stay at the current control state, or to move to other control states and to which ones. As usual, the strategy should recommend only moves that are possible as determined by the values of the continuous variables and transition guards.

We study linear time specifications given as LTL formulae, or more generally QPLTL formulae. Such a specification dictates the desired subset of infinite control state sequences. A strategy is winning with respect to an LTL or QPLTL formula φ iff every infinite control state sequence generated by the controlled plant satisfies φ. By a controller, we mean a winning strategy.

We show that: if there is *no* delay associated with sensing, then the controller synthesis problem for LTL specifications is decidable. Further, if a controller exists, then we can effectively synthesize one in the form of a (finite) RDA. These results also hold for QPLTL specifications. We emphasize that though sensing delays are prohibited, update delays are allowed.

We do not know how to settle the controller synthesis problem for RDAs when sensing delays are present. The key obstacle is that in such case, a strategy has *incomplete* information about the variables of the RDA.

Parts of the results on distributed controller synthesis were joint work with P. Madhusudan and P. S. Thiagarajan, and were published as [52]. Parts of the results on synthesis of admission controllers for real-time systems with tasks were jointly obtained with P. S. Thiagarajan and Wang Yi. The regularity result for RDAs is closely related to the joint work [2] with Manindra Agrawal, Frank Stephan and P. S. Thiagarajan.

1.4. Thesis Organization

In the next chapter, we review some preliminaries of automata and logics over infinite words and infinite trees. We also give a precise formulation of a basic controller synthesis problem in a sequential setting.

In chapter 3, we investigate the distributed controller synthesis for CCP plants. We prove that the MSO theory of the event structure unfolding of every CCP is decidable. Using this logical result, we obtain decidability results of distributed controller synthesis for CCP plants for both robust linear time specifications and branching time specifications. In both cases, we show further that, if a distributed controller exists, then we can effectively synthesize a finite state one. On the negative side, we show that the distributed controller synthesis problem for CCP plants is undecidable for linear time specifications that are allowed to be non-robust. We also show that the strict distributed controller synthesis problem for CCP plants is undecidable for linear time specifications, even if they are robust.

In chapter 4, we study the synthesis of admission controllers for real-time systems with tasks. We prove that, given a task plant based on timed automata extended with tasks and a QoS requirement in LTL or QPLTL, we can effectively determine whether there exists an admission controller. Further, in case such an admission controller exists, then we can effectively synthesize one in the form of a (finite) timed automaton.

In chapter 5, we consider controller synthesis in hybrid settings. We show that the language of control state sequences of an RDA is regular. Using this regularity result, we prove that, if there is no sensing delay, then the controller synthesis problem for RDAs is decidable for LTL and QPLTL specifications. Further, if a controller exists, then we can effectively synthesize one in the form of a (finite) RDA.

In the concluding chapter, we discuss prospects of future directions.

CHAPTER 2

Automata, Logics, Controller Synthesis

In this chapter, we review basic materials of automata over infinite words and infinite trees in section 2.1, and logics over infinite words and trees in section 2.2. The purpose is mainly to fix notations and terminologies. The tools in section 2.1 and 2.2 will be used in the next three chapters in one way or another. Finally, in section 2.3, we give a formulation of a basic controller synthesis problem in sequential settings with linear time specifications. This is just to illustrate the various notions of controller synthesis in a precise manner.

2.1. Automata on Infinite Words and Infinite Trees

Here we review automata running over infinite words and infinite trees. We shall need only automata with Büchi and Rabin acceptance conditions. For a detailed reference, we recommend [73].

In what follows, we fix Σ to be a finite alphabet. Let Σ^ω denote the set of infinite words (ω-words) over Σ. A *non-deterministic Büchi automaton* over Σ is a structure $\mathcal{B} = (Q, q_{in}, \Sigma, \hookrightarrow, F)$ where Q is a finite set of states, $q_{in} \in Q$ the initial state, $\hookrightarrow \subseteq Q \times \Sigma \times Q$ the transition relation and $F \subseteq Q$ the set of accepting states. Let $\sigma = a_0 a_1 \ldots$ be in Σ^ω. A *run* of $\mathcal{B}$ over σ is an infinite sequence $\rho = q_0 q_1 \ldots$, where $q_i \in Q$ for $i = 0, 1, \ldots$, such that $q_0 = q_{in}$, and $q_i \overset{a_i}{\hookrightarrow} q_{i+1}$ for $i = 0, 1, \ldots$. The run ρ is *accepting* iff for some $\hat{q} \in F$, $\hat{q}$ occurs in ρ infinitely often, that is, there exist infinitely many i with $q_i = \hat{q}$. We say σ is accepted by $\mathcal{B}$ iff there exists an accepting run of $\mathcal{B}$ over σ. The *language* of $\mathcal{B}$ is the set of ω-words over Σ that are accepted by $\mathcal{B}$. We say $\mathcal{B}$ is *deterministic* iff for each $s \in S$, $a \in \Sigma$, there is at most one $s' \in S$ with $s \overset{a}{\hookrightarrow} s'$.

A *non-deterministic Rabin automaton* over Σ is a structure $\mathcal{R} = (Q, q_{in}, \Sigma, \hookrightarrow, \mathcal{F})$ where Q, q_{in}, $\hookrightarrow$ are as those of a Büchi automaton, while $\mathcal{F} = \{(E_1, F_1), (E_2, F_2), \ldots, (E_k, F_k)\}$ is a set of accepting pairs, where E_i, F_i are subsets of Q. Let σ be in Σ^ω. The notion of ρ in Q^ω being a run of $\mathcal{R}$ over σ is defined in the same way as for Büchi automata. However, we say ρ is *accepting* iff for some accepting pair (E_ℓ, F_ℓ) in $\mathcal{F}$, it is the case that every state in E_ℓ occurs in ρ only finitely often, while some state in F_ℓ occurs in ρ infinitely often. More precisely, we say a state $\hat{q}$ occurs in $\rho = q_0 q_1 \ldots$ finitely often iff there exists i in $\{0, 1, \ldots\}$ such that $q_j \neq \hat{q}$ for every $j > i$.

As usual, we say $\mathcal{R}$ accepts σ iff there exists an accepting run of $\mathcal{R}$ over σ. The language of $\mathcal{R}$ is defined in the obvious way. We define *deterministic* Rabin automata in the same way as for deterministic Büchi automata. We also note that a non-deterministic Büchi automaton can be viewed as a non-deterministic Rabin automaton in the obvious way.

Languages accepted by non-deterministic Büchi automata are called ω-regular languages. By a regular subset of Σ^ω, we shall mean an ω-regular language over Σ. It is known that the class of languages accepted by non-deterministic Rabin automata and the class of languages accepted by deterministic Rabin automata are the same and are both equal to the class of ω-regular languages. However, there exist ω-regular languages that can not be accepted by any deterministic Büchi automaton.

Next we review infinite trees and automata running over Σ-labelled infinite trees. We fix a finite alphabet Γ in what follows. Let $\Gamma^\star$ denote the set of (finite) words over Γ. A Γ-tree is a prefix-closed regular subset of $\Gamma^\star$. Elements of T are *nodes* with ε being the *root*. In particular, we call $\Gamma^\star$ the full Γ-tree. We shall define tree automata with respect to Γ-trees. This differs from standard treatment of tree automata in the literature which typically deals with only the full Γ-tree ([73]). However, one can easily see that our definition involves no loss of generality.

Let T be a Γ-tree. For a node w in T, we define the set of *successors* of w, denoted $Succ_T(w)$, to be the set $\{wv \in T \mid v \in \Gamma\}$. We will implicitly assume the Γ-trees we encountered are such that every node has a nonempty set of successors. A *path* of T is a subset $\pi \subseteq T$ satisfying that $\varepsilon \in \pi$ and every node in π has exactly one successor in π. Note that a path must be an infinite set of nodes. Abusing notation, we will often write the path π as the infinite sequence $d_0 d_1 \ldots$ in Γ^ω in the sense that the set of finite prefixes of $d_0 d_1 \ldots$ is precisely π. The *direction* of a node w, denoted $dir(w)$, is defined as follows. $dir(\varepsilon)$ is a special element $\$ \notin \Gamma$. For $wv \in T$, where $v \in \Gamma$, we set $dir(wv) = v$.

A Σ-labelled Γ-tree is a pair (T, η), where T is a Γ-tree and $\eta : T \to \Sigma$ a labelling function. We say T is the underlying tree of (T, η). In what follows, we fix Γ and a Γ-tree T.

A *non-deterministic Büchi tree automaton* $\mathcal{B}$ over Σ-labelled Γ-trees (whose underlying tree is T) is a structure $(Q, q_{in}, \Sigma, \hookrightarrow, F)$ where Q is a finite set of states and $q_{in} \in Q$ the initial state. For $\Gamma' \subseteq \Gamma$, let $Fun(\Gamma', Q)$ denote the set of functions from Γ' to Q. The transition relation $\hookrightarrow$ is a subset of $Q \times \Sigma \times \bigcup_{\Gamma' \subseteq \Gamma} Fun(\Gamma', Q)$. Lastly, $F \subseteq Q$ is the set of accepting states.

Let (T, η) be a Σ-labelled Γ-tree. A run of $\mathcal{B}$ over the (T, η) is a Q-labelled Γ-tree (T, ρ) which satisfies:

- $\rho(\varepsilon) = q_{in}$.
- For every node w in T, there exists a transition $q \overset{a}{\hookrightarrow} \chi$ such that $q = \rho(w)$, $a = \eta(w)$ and χ is a function from the set $\{dir(w') \mid w' \in Succ_T(w)\}$ to Q which satisfies: for each w' in $Succ_T(w)$, we have $\rho(w') = \chi(dir(w'))$.

Intuitively, if $\mathcal{B}$ is at state q while encountering a node v in T, then $\mathcal{B}$ reads the label of v (dictated by η) and propagates a copy of itself to the successor nodes of v simultaneously. The run (T, ρ) is *accepting* iff for every path $d_0d_1\ldots$ in T, there exists a state $\hat{q} \in F$ which occurs infinitely often in $q_0q_1\ldots$, where $q_i = \rho(d_0d_1\ldots d_i)$ for $i = 0, 1, \ldots$. We say (T, η) is accepted by $\mathcal{B}$ iff there exists an accepting run of $\mathcal{B}$ over (T, η). By the *language* of $\mathcal{B}$, we mean the set of Σ-labelled Γ-trees (whose underlying tree is T) that are accepted by $\mathcal{B}$.

We say the Büchi tree automaton $\mathcal{B}$ is *deterministic* iff for every $q \in Q$, $a \in \Sigma$, there exists at most one χ in $\bigcup_{\Gamma' \subseteq \Gamma} Fun(\Gamma', Q)$ with $q \overset{a}{\hookrightarrow} \chi$.

A *non-deterministic Rabin tree automaton* $\mathcal{B}$ over Σ-labelled Γ-trees (whose underlying tree is T) is a structure $(Q, q_{in}, \Sigma, \hookrightarrow, \mathcal{F})$ where Q, q_{in}, $\hookrightarrow$ are as those for non-deterministic Büchi tree automata, while $\mathcal{F} = \{(E_1, F_1), (E_2, F_2), \ldots, (E_k, F_k)\}$ is a set of accepting pairs, where E_i, F_i are subsets of Q.

As expected, runs of $\mathcal{R}$ over an input tree (T, η) are defined in the same way as non-deterministic Büchi tree automata. However, we say the run (T, ρ) is *accepting* iff every path $d_0d_1\ldots$ of T satisfies the following property: for some accepting pair (E_ℓ, F_ℓ) in $\mathcal{F}$, we have that every state in E_ℓ occurs only finitely often in $q_0q_1\ldots$, where $q_i = \rho(d_0d_1\ldots d_i)$ for $i = 0, 1, \ldots$, while some state in F_ℓ occurs in $q_0q_1\ldots$ infinitely often. As usual, we say $\mathcal{R}$ accepts the input tree (T, η) iff there exists a run of $\mathcal{R}$ over (T, η). The *language* of $\mathcal{R}$ is defined in the usual way.

Deterministic Rabin tree automata are defined in the same way as for deterministic Büchi tree automata. We also note that a non-deterministic Büchi tree automaton can be trivially viewed as a non-deterministic Rabin tree automaton.

It is known that non-deterministic Rabin tree automata and deterministic Rabin tree automata have the same expressive power. In other words, given a non-deterministic Rabin tree automaton $\mathcal{R}$, there exists a deterministic Rabin tree automaton $\mathcal{R}'$ such that $\mathcal{R}$ and $\mathcal{R}'$ accepts the same set of trees. However, non-deterministic Büchi tree automata is strictly less expressive than non-deterministic Rabin tree automata.

Two Σ-labelled Γ-trees are said to be *isomorphic* iff there exists a bijective mapping between the nodes such that the labels are preserved. Suppose (T, η) is a Σ-labelled Γ-tree. Let w be a node. The subtree of (T, η) rooted at w, denoted (T_w, η_w), is given by: $T_w = \{u \mid wu \in T\}$ and $\eta_w(u) = \eta_w(wu)$. We say (T, η) is *regular* iff it has finitely many isomorphic subtrees.

By Rabin's tree theorem [63], given the Rabin tree automaton $\mathcal{R}$ over Σ-labelled Γ-trees (whose underlying tree is T), one can effectively determine whether the language of $\mathcal{R}$ is nonempty. Moreover, if the answer is positive, then the nonemptiness testing algorithm also produces a *regular* Σ-labelled Γ-tree (T, η) that is accepted by $\mathcal{R}$.

2.2. Logics over Infinite Words and Infinite Trees

In this section, we introduce logics over infinite words and infinite trees. We shall need only LTL (linear time temporal logics) and QPLTL (quantified propositional LTL) over infinite computation sequences, and the monadic second order (MSO) logics over infinite trees. For detailed references, we recommend [21] for LTL and QPLTL, and [73] for MSO logics over infinite trees.

In what follows, we fix a finite set of atomic propositions AP. The set of LTL formulae over AP, denoted $LTL(AP)$, is defined inductively as follows:

- If $p \in AP$, then p is in $LTL(AP)$.
- If ψ, ψ' are in $LTL(AP)$, then so are $\sim \psi$, $\psi \vee \psi'$, $\mathcal{X}(\psi)$, and $\psi \, \mathcal{U} \, \psi'$.

Intuitively, $\mathcal{X}$ stands for "next" and $\mathcal{U}$ "until". Common derived operators $\Diamond$ ("future") and $\Box$ ("globally") can be defined as: $\Diamond\varphi = \mathit{true}\,\mathcal{U}\,\varphi$; and $\Box\varphi = \sim(\Diamond(\sim\varphi))$.

Models for $LTL(AP)$ are infinite sequences over 2^{AP}. Let $\sigma = \alpha_0\,\alpha_1\,\ldots$ be in $(2^{AP})^\omega$. Set $\sigma(i) = \alpha_i$ for $i = 0, 1, \ldots$. The notion that the LTL formula ψ being satisfied by σ at position i, denoted $\sigma, i \models \psi$, is defined inductively as follows:

- $\sigma, i \models p$ iff $p \in \sigma(i)$.
- $\sigma, i \models \sim\psi$ iff it is *not* the case that $\sigma, i \models \psi$.
- $\sigma, i \models \psi \vee \psi'$ iff $\sigma, i \models \psi$ or $\sigma, i \models \psi'$.
- $\sigma, i \models \mathcal{X}(\psi)$ iff $\sigma, i+1 \models \psi$.
- $\sigma, i \models \psi\,\mathcal{U}\,\psi'$ iff there exists $j > i$ such that $\sigma, k \models \psi$ for every k with $i \leq k < j$ and $\sigma, j \models \psi'$.

Now we say that σ is a model of ψ iff $\sigma, 0 \models \psi$.

The *size* of a formula ψ in $LTL(AP)$ is denoted $|\psi|$ and is defined inductively as follows:

- $|p| = 1$ for $p \in AP$.
- $|\sim\psi| = 1 + |\psi|$ and $|\psi \vee \psi'| = |\psi| + |\psi'| + 1$.
- $|\mathcal{X}(\psi)| = 1 + |\psi|$ and $|\psi\,\mathcal{U}\,\psi'| = |\psi| + |\psi'| + 1$.

We note that ([78]), given a formula ψ in $LTL(AP)$, one can effectively construct a non-deterministic Büchi automaton $\mathcal{B}_\psi$ over 2^{AP} with the following property: for every σ in $(2^{AP})^\omega$, σ is accepted by $\mathcal{B}_\psi$ iff σ is a model of ψ. And $\mathcal{B}_\psi$ will have $2^{O(|\psi|)}$ states.

The set of QPLTL formulae over AP, denoted $QPLTL(AP)$, is defined inductively as follows:

- If $p \in AP$, then p is in $QPLTL(AP)$.
- If ψ, ψ' are in $QPLTL(AP)$, then so are $\sim\psi$, $\psi \vee \psi'$, $\mathcal{X}(\psi)$, and $\psi\,\mathcal{U}\,\psi'$.
- If φ is in $QPLTL(AP)$ and $p \in AP$, then $\exists p.\ \varphi$ is in $QPLTL(AP)$.

Thus *QPLTL*(AP) is a proper superset of *LTL*(AP). As with LTL, models for QPLTL are infinite sequences over 2^{AP}. Let $\sigma = \alpha_0\,\alpha_1\,\ldots$ be in $(2^{AP})^\omega$. Set $\sigma(i) = \alpha_i$ for $i = 0, 1, \ldots$. The notion that the QPLTL formula ψ being satisfied by σ at position i, denoted $\sigma, i \models \psi$, is defined inductively as follows:

- The cases of p, $\sim \psi$, $\psi \vee \psi'$, $\mathcal{X}(\psi)$, $\psi\,\mathcal{U}\,\psi'$ are defined in the same way as *LTL*(AP).
- $\sigma, i \models \exists p.\ \psi$ iff there exists σ' in $(2^{AP})^\omega$ such that $\sigma', i \models \psi$ and σ' differs from σ in at most the truth value of p. More precisely, let $\sigma' = \alpha'_0\,\alpha'_1\,\ldots$ with $\sigma'(i) = \alpha'_i$ for $i = 0, 1, \ldots$, then for every $i = 0, 1, \ldots$, for every $q \in AP$, q is in $\sigma(i)$ iff q is in $\sigma'(i)$.

It is known that QPLTL is strictly more expressive than LTL [80]. For example, the QPLTL formula

$$\exists q\ (q \wedge \mathcal{X}(\sim q) \wedge \Box(q \rightarrow \mathcal{X}(\mathcal{X}(q))) \wedge \Box(q \rightarrow p))$$

asserts that p holds at all even indices, while p may or may not hold at odd indices. In general, for a fixed integer $n > 1$, one can construct a QPLTL formula Φ_n, which asserts the property that p holds at all indices that are multiples of n, while p may or may not hold at other indices. The formula Ψ_n will quantify over $\lceil \log_2 n \rceil$ atomic propositions and use them to "count" periodically from 0 to $n-1$. It can be proved [80] that for any $n > 1$, the Ψ_n is asserting can *not* be expressed in LTL, that is, there is no formula ψ in *LTL*(AP) such that the set of models of ψ is equal to that of Ψ_n.

It is known that QPLTL has the same expressive power as the class of ω-regular languages [21]. In other words, for any ω-regular language L over 2^{AP}, one can effectively construct a formula in *QPLTL*(AP) such that L is precisely the set of models of *QPLTL*(AP). Conversely, for any formula ψ in *QPLTL*(AP), one can effectively construct a non-deterministic Büchi automaton $\mathcal{B}$ over 2^{AP} such that the language of $\mathcal{B}$ is precisely the set of models of ψ.

In what follows, we fix a finite alphabet Σ. We next introduce the monadic second order (MSO) logic of n-successors ($n = |\Sigma|$) interpreted over the full Σ-tree $TR = \Sigma^\star$, denoted $MSO(\Sigma)$. The syntax is given by:

$$MSO(TR) ::= succ_a(x, y) \mid x \in X \mid \exists x\,(\varphi) \mid \exists X(\varphi) \mid \sim \varphi \mid \varphi' \vee \varphi' \ ,$$

where a ranges over Σ. As usual, $x, y, \ldots$ are individual variables and $X, Y, \ldots$ are set variables. An *interpretation* of TR assigns to every individual variable a member of $\Sigma^\star$ and to every set variable a subset of $\Sigma^\star$. For an interpretation $\mathcal{I}$ of TR, we have $TR \models_{\mathcal{I}} succ_a(x, y)$ iff $\sigma a = \sigma'$ where $\sigma = \mathcal{I}(x)$, $\sigma' = \mathcal{I}(y)$. With this, the semantics of $MSO(TR)$ is clear ([73]). As usual, sentences are formulae that do not have free individual or set variables. By the MSO theory of TR, we shall mean the set of sentences in $MSO(TR)$ that evaluate to true in TR.

Rabin's famous result [63] states that the MSO theory of 2-successors is decidable. It follows easily that the MSO theory of n-successors interpreted over TR is decidable. That is, given any sentence φ in $MSO(TR)$, we can effectively determine whether φ is true. This forms the foundation for model checking [17] and controller synthesis problems in sequential settings.

The key ideas for establishing the decidability of $MSO(TR)$ are as follows. Firstly, models of formulae can be view as certain labelled trees. Secondly, for a formula φ in $MSO(TR)$, one can effectively construct a non-deterministic Rabin tree automaton $\mathcal{R}$ which accepts precisely the set of models of φ. Finally, by Rabin's tree theorem [63], we can effectively test whether the language accepted by a tree automaton is nonempty.

2.3. Controller Synthesis

In this section, we give a formal introduction to controller synthesis in a basic sequential setting where the plant model is based on a finite transition system and the specification is an LTL formula.

A *plant* $\mathcal{A}$ is a structure $(Q_e, Q_s, q_{in}, \longrightarrow, AP, \lambda)$, where Q_e, Q_s are disjoint finite sets of *environment states* and *system states*. $q_{in} \in Q_e$ is the initial state. $\longrightarrow \subseteq (Q_e \times Q_s) \bigcup (Q_s \times Q_e)$ the transition relation. AP is a set of atomic propositions, and $\lambda : \{Q_e \cup Q_s\} \rightarrow 2^{AP}$ is a labelling function that maps each environment or system state to a subset of atomic propositions. Intuitively, $\mathcal{A}$ describes the possible interactions of an open system against its environment, where for each state s, the set $\lambda(s)$ represents atomic propositions that are true in s. Figure 2.1 shows a plant, where environment states indicated by circles and system states drawn as boxes. The inscription of each state s is the set of atomic propositions $\lambda(s)$.

A *specification* is an LTL formula ψ over AP. In what follows, we fix the plant $\mathcal{A}$ and the specification ψ.

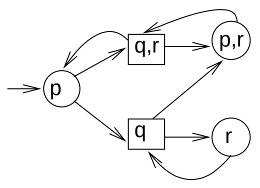


FIGURE 2.1. A plant

For a state $q \in Q_e$, we define $Move(q) = \{q' \in Q_s \mid q \longrightarrow q'\}$. In other words, $Move(q)$ is the set of possible moves that the environment may take at state q. Similarly, for $q \in Q_s$, we define $Move(q) = \{q' \in Q_e \mid q \longrightarrow q'\}$. Intuitively, $Move(q)$ is the set of moves available to the system at q. Without loss of generality, we will assume $Move(q) \neq \emptyset$ for every $q \in Q_e \cup Q_s$.

A *play* of $\mathcal{A}$ is a finite sequence $q_0 q_1 \ldots q_n$ over $Q_e \cup Q_s$, such that $q_0 = q_{in}$ and $q_i \longrightarrow q_{i+1}$ for $i = 0, \ldots, n-1$. We let $Play(\mathcal{A})$ denote the set of plays of $\mathcal{A}$. We are now ready to define strategies. A *strategy* for $\mathcal{A}$ is a function $f : \mathit{Play}(\mathcal{A}) \rightarrow 2^{Q_e \cup Q_s}$ such that for every play $\rho = q_0 q_1 \ldots q_n$, we have:

- If $q_n \in Q_e$, then $f(\rho) = \mathit{Move}(q_n)$.
- If $q_n \in Q_s$, then $f(\rho) \subseteq \mathit{Move}(q_n)$.

The first condition states that f does not restrict the environment's moves in any way. The second condition demands that f only recommends moves among the structurally possible ones indicated by the plant.

The notion of a play being *according to* a strategy f is defined inductively as follows:

- ε is according to f.
- If ρ is according to f and $q \in f(\rho)$, then $\rho\, q$ is according to f.

We say the strategy f is *non-blocking* iff every play according to f can be extended to a longer one that is also according to f. Note that our notion of non-blocking is different from and in fact weaker than that of supervisory control of discrete event systems studied in the control community ([67]).

An infinite play of $\mathcal{A}$ is an infinite sequence ρ over Q such that every finite prefix of ρ is a play of $\mathcal{A}$. The infinite play ρ is said to be according to a strategy f iff every finite prefix of ρ is according to f.

Let $\rho = q_0\, q_1 \ldots$ be an infinite play. We say ρ is a model of ψ iff the infinite sequence $\lambda(q_0)\, \lambda(q_1) \ldots$ over 2^{AP} is a model of ρ. We say the strategy f is *ψ-winning* iff f is non-blocking and every infinite play according to f is a model of ψ.

The sequential controller synthesis problem can now be stated: Given the pair $(\mathcal{A}, \psi)$, where $\mathcal{A}$ is a plant and ψ is a specification, can one effectively determine whether there exists a ψ-winning strategy for $\mathcal{A}$?

The following result is well-known in the literature (for instance, see [14, 74]).

Proposition 2.1. *Given the pair $(\mathcal{A}, \psi)$, where $\mathcal{A}$ is a plant and ψ is a specification, one can effectively determine whether there exists a ψ-winning strategy.*

Further, if the answer is positive, then one can effectively construct a finite state *ψ-winning strategy $\hat{f}$ presented in the form of a finite transition system $\mathcal{C}$. And the parallel composition of $\mathcal{C}$ and $\mathcal{A}$ will produce only infinite plays according to $\hat{f}$.*

Instead of LTL, one can also consider a specification L to be an ω-regular language over $Q_e \cup Q_s$. Such a specification L may be presented as a non-deterministic Büchi automaton. We define that a strategy f is winning for L iff f is non-blocking and every infinite play according to f is in L. We remark that proposition 2.1 also holds if the specification is an ω-regular language over $Q_e \cup Q_s$, instead of an LTL formula.

CHAPTER 3

Distributed Controller Synthesis for Connectedly Communicating Processes (CCPs)

The subject of this chapter is controller synthesis in distributed settings. We are mainly interested in distributed controller synthesis problems associated with a subclass of distributed systems which we called connectedly communicating processes (CCPs). Section 3.1 gives an overview of the CCP model and our results. Subsequently, we present related work in section 3.2. In section 3.3, we formulate the CCP model based on asynchronous transition systems. As the foundation for distributed controller synthesis, we prove, in section 3.4 that the MSO (monadic seconder order) theory of the event structure unfolding of every CCP is decidable, where the event structure unfolding of a CCP represents its non-interleaved branching time behaviour. We note that this logical result is also of independent interest for verification of distributed systems that can be modelled as CCPs.

We next formulate a model of distributed plants based on CCPs, in section 3.5. We then show, in section 3.6, that the distributed controller synthesis problem for CCP plants is decidable for robust linear time specifications and branching time specifications given as formulae in the MSO logic of the event structure unfolding of the CCP plant. By a robust linear time specification, we mean one that does not discriminate between two different linearizations of the same partially ordered execution. For both kinds of specifications, we prove further in section 3.7 that, if a distributed controller exists, then a finite state one can be effectively synthesized as a CCP.

On the negative side, we show in section 3.8 the distributed controller synthesis problem with CCP plants is undecidable for linear time specifications that are allowed to be non-robust. In addition, we also show that the strict distributed controller synthesis problem with CCP plants is undecidable for linear time specifications, even if they are robust.

We conclude with prospects of future directions in section 3.9.

3.1. Overview

Informally, the distributed controller synthesis problem is: Given a distributed plant and a specification of desired behaviour, determine whether there exist a family of local strategies, one for each component of the distributed plant, such that the collective controlled behaviour satisfies the specification. The problem have been studied in the literature under several different frameworks, varying mainly according to the model of the distributed plant, the kind of specifications and the type of local strategies. We follow the framework of modelling the distributed plant using asynchronous transition systems and that the local strategies are view-based. And we study linear time and branching time specifications. In what follows, we make precise our framework and outline our results. In the next section, we will discuss in details related work in our framework and in various other frameworks.

A *distributed plant* is a family of communicating sequential open reactive systems (which we called processes), each of which interacts with its local environment. We shall model a distributed plant based on a (finite) asynchronous transition system, which consists of a family of sequential transition systems that communicate by synchronizing on common actions. If an action a involves a subset of processes P, then a is enabled only when every process in P is ready to execute a. A *linear time specification* is an

ω-regular language over the action alphabet of the distributed plant. Later we will also discuss branching time specifications.

A *local strategy* for process p controls the execution of p by restricting, at each stage of computation, the possible moves of p. It does so based on the *local view* of the process p which consists of the history of actions executed by p as well as actions executed by other processes that p comes to know via synchronization, directly or indirectly. The local strategy for process p must not restrict in any way the moves of the local environment of p. A synchronization action involving a subset of P of processes can be performed only when it is permitted by all the local strategies of the processes in P.

A family of local strategies, one for each process, is winning for a linear time specification iff the infinite runs generated by the collective controlled behaviour fall within the linear time specification. A distributed controller is a winning family of local strategies.

We also demand that a family of local strategies, one for each process, is non-blocking in the sense that the distributed plant will not deadlock by following the local strategies. This does not rule out the possibility that some (but not all) processes may become deadlocked. However, to demand that every process will not deadlock, one can place appropriate liveness conditions in the specification. For instance, we can assert that actions of each process must occur infinitely often.

As mentioned in section 2.3, to solve controller synthesis for sequential systems with respect to even linear time specifications, one has to study the branching time behaviour of sequential systems. This is mainly due to that the environments' moves can not be restricted in any way by a strategy. A sequential system can be modelled by a transition system. The branching time behaviour of a transition system is defined by its tree unfolding. By the MSO (monadic second order) logic of a transition system, we mean the MSO logic of n-successors interpreted over the tree unfolding of the transition system. The foundation for solving sequential controller synthesis is the logical result that the MSO theory of every transition system is decidable, which follows from Rabin's famous theorem [63] of the decidability of the MSO theory of 2-successors.

To study the distributed controller synthesis problem, we set out to investigate the *non-interleaved* branching time behaviour of asynchronous transition systems. The non-interleaved branching time behaviour of an asynchronous transition system is captured by its event structure unfolding. The event structure consists of events which represent "execution points" of the asynchronous transition system, and the causality relation (a partial order) as well as the conflict relation between events. One can also define a natural MSO logic for event structures in which the causality relation and the conflict relation are the non-logical predicates and quantification is carried out over individual and subsets of events. By the MSO logic of an asynchronous transition system $\mathcal{A}$, we shall mean the MSO logic of the event structure unfolding of $\mathcal{A}$. However, it is *not* the case that the MSO theory of every asynchronous transition system is decidable. In fact, one can easily construct a simple asynchronous transition system whose MSO theory is undecidable. Hence, a logical question of fundamental interest to distributed controller synthesis is: What is the precise subclass of asynchronous transition systems for which the MSO theories are decidable?

We provide a partial answer to this logical question by identifying the subclass of *connectedly communicating* asynchronous transition systems and proving that the MSO theory of every connectedly communicating asynchronous transition system is decidable. As the name suggests, the connectedly communicating criterion requires processes to communicate with each other frequently. More precisely, we say an asynchronous transition system is connectedly communicating iff there exists a bound k such that for every process p, q, if process p executes k steps without hearing from process q

either directly or indirectly and reaches a state s, then starting from s it will never hear from q again, directly or indirectly. We note that for a given asynchronous transition system, one can effectively determine whether it is connectedly communicating. We shall refer to connectedly communicating asynchronous transition systems simply as connectedly communicating processes (CCP). CCPs can model naturally distributed protocols where processes communicate frequently with each other so that they maintain bounded loss of status on each other. Further, if the loss of process p on the status of q exceeds the given bound, then p will never obtain any further information about q.

Our technique of establishing the decidability of the MSO theories of CCPs consists of extracting a regular tree from the event structure induced by a CCP with the nodes of this tree corresponding to the events of the event structure such that the causality relation are definable in the MSO theory of trees. This representation is obtained directly and broadly preserves the structure of the event structure. As might be expected, the decision procedure for determining the truth of a sentence φ in the MSO logic of a CCP is non-elementary in the size of φ.

Using our logical result that the MSO theory of every CCP is decidable, we establish the decidability of the distributed controller synthesis problem for distributed plants based on CCPs with respect to *robust* linear time specifications. By a *robust* specification, we mean one that does not discriminate two different interleavings of the same partially ordered execution. In other words, a robust specification is an ω-regular (Mazurkiewicz) trace language ([18]). More precisely, we prove that, given a CCP plant and a robust linear time specification, one can effectively determine whether there exists a distributed controller. This proof is by constructing a sentence in the MSO theory of the given CCP plant, that asserts the existence of a distributed controller. We note that the complexity of this decision procedure is non-elementary in the size of the linear time specification. Further, we show that if a distributed controller exists, then we can effectively synthesize one that consists of a family of *finite state* local strategies. Such a finite state distributed controller can be collectively represented also as a CCP.

In fact, the connectedly communicating criterion that we impose on distributed plants and the robustness requirement that we demand on linear time specifications are motivated from undecidability results in distributed controller synthesis. Many undecidability results in distributed controller synthesis rely on the undecidability of *multi-player games with incomplete information* studied in [60]. There are two key ingredients for establishing this undecidability result. Firstly, the local players (in our case, processes) may not synchronize frequently and hence can have an unbounded loss of information on the status of each others' executions. Secondly, the linear time specification is allowed to an arbitrary ω-regular language, which may discriminate different linearizations of the same partially ordered execution. Our connectedly communicating criterion ensures that processes synchronize frequently so that the loss of information on the status of each others' executions remain bounded. The robustness property we placed on the linear time specifications prohibits specifications to discriminate different linearizations of the same partially ordered execution.

Using our decidability result of the MSO theories of CCPs, we also investigate the distributed controller synthesis problem for distributed plants based on CCPs with respect to branching time specifications. We consider a branching time specification to be given by a formula in the MSO logic of the CCP plant. We say a family of local strategies is winning for such an MSO formula iff the "sub-event structure" resulting from the overall controlled branching time behaviour of the distributed plant satisfies the MSO formula. We show that, given a CCP plant and a branching time specification, one can effectively determine whether there exists a distributed controller. We note that the complexity of this decision procedure is non-elementary in the size of the branching time specification. Further, in case a distributed controller exists, then a finite state one can be effectively synthesized in the form of a CCP.

On the negative side, we first show that the distributed controller synthesis problem is undecidable for linear time specifications that are allowed to be non-robust, even if the distributed plant is based on a CCP.

We show also a negative result concerning distributed controller synthesis with regard to *strictly local strategies*. In contrast to a local strategy, we demand that a strictly local strategy for p should recommend moves for p based *only* on knowledge of actions executed by p. In linear time setting, the *strict distributed controller synthesis problem* is: given a distributed plant and a linear time specification, can one determine whether there exists a family of strictly local strategies such that the infinite runs generated by collectively controlled plant fall within the linear time specification. We prove that the strict distributed controller synthesis problem for CCP plants is undecidable for linear time specifications, even if they are robust. It would be interesting to also study the strict distributed controller problem with respect to branching time specifications.

3.2. Related Work

As for related work, we first discuss literature related to our logical result that the MSO theory of every CCP is decidable. A variety of branching time logics based on event structures have been proposed in the literature (see for instance [59] and the references therein). The objectives of these logics are to extend classical branching time temporal logics over trees (for example, CTL, CTL* [21]) to event structures that admit efficient model checking procedures. Since these logics do not deal directly with the MSO theories over event structures, they are in general not expressive enough for solving distributed controller synthesis associated with asynchronous transition systems.

The work [47] investigates the MSO theories over (regular) event structures. In our terms, the result in [47], that is closely related to our logical result, can be stated as follows: the MSO theory of every asynchronous transition system is decidable *provided* set quantification is restricted to *conflict-free* subsets of events. It is however difficult to exploit this result to solve distributed controller synthesis problems associated with asynchronous transition systems. For, as we will see in section 3.6, the set of events corresponding to a family of local strategies will in general contain events that are in conflict, where these events correspond to the different choices available to the local environments. This is due to that the local environments' moves can not be restricted in any way by the local strategies.

We now turn to more directly related work on distributed controller synthesis. This problem has been studied in both computer science and the control theory community.

In computer science, there are two main frameworks under which one studies distributed controller synthesis. The first line of work considers a distributed plant to be an *architecture*, which consists of a set of local *sites* that communicate via fixed channels; and the other, and in addition the local sites communicate via fixed channels with local environments. To be precise, in this setting, the literature studies both the distributed realizability problem and the distributed controller synthesis problem. In this setting, the distributed realizability problem is: Given an architecture and a linear time or branching time specification, does there exist a collection of local programs one for each local site, such that the overall behaviour of architecture satisfies the specification? The distributed controller synthesis problem is: Given an architecture *and* a collections of local programs, one for each local site, and a (linear time or branching time) specification, does there exist a collection of local strategies, one for each local site that will restrict the corresponding local program, such that the overall controlled behaviour of the local programs satisfies the specification?

The second line of work in distributed controller synthesis, to which our work belongs, models distributed plants based on asynchronous transition systems in which processes communicate via synchronization on common actions.

We begin by reviewing the first line of work in distributed controller synthesis. The work [62] studies the distributed realizability problem with the notion of architecture. It is reported in [62] that for linear time specifications, the distributed realizability problem is undecidable even for the simple architecture consisting of just two sites that do not have any communication channels between them. This undecidability result has its root in the undecidability of multi-player games with incomplete information studied in [60]. It was also shown in [62] that the distributed realizability problem for linear time specifications is decidable for a small class of architectures, called hierarchical architectures, in which the local sites are linearly ordered and information flows in one direction.

The work [39] studied the distributed realizability problem with the notion of architecture but for branching time specifications. The main result of [39] states that the distributed realizability problem for branching time specifications given as CTL* formulas is decidable for the subclass of architectures in which there is a linear or cyclic order among the local sites and information flows in either one or both directions. This subclass of architectures properly includes the subclass of hierarchical architectures studied in [62].

The work [49] studied the distributed controller synthesis under the framework of architecture but consider a specification to be given as a conjunction of local linear time specifications, one for each local site. It is shown in [49] the distributed controller synthesis problem for local specifications is decidable for the class of architectures such that each connected component is either a *clean pipeline*, or is a sub-architecture of a *clean pipeline*.

The work of [79] also studies distributed realizability problem under the framework that processes communicate via fixed channels, albeit in a more abstract form. The main contribution of [79] are two theorems that allow to simplify the distributed realizability problem by reducing the number of players (processes) and the amount of nondeterminism of environments' moves. In some cases, these two theorems allow one to reduce the distributed realizability to the setting that there is only one single player or the setting that that the environments are deterministic. The former is then the realizability problem in sequential settings, while the latter is also shown in [79] to be effectively solvable.

Next we review literature in the second line of work in distributed controller synthesis. These include [25, 26, 48, 51].

In contrast with our work, [51] obtains decidability in distributed controller synthesis for general distributed plants but imposes restrictions on local strategies. The main result of [51] states that: given *any* distributed plant based on asynchronous transition systems and a *robust* specification, one can effectively determine whether there exists a winning family of *clocked* and *com-rigid* strategies. And further, if one drops any (or more of) the three restrictions, namely the specification is robust, local strategies are clocked, local strategies are com-rigid, then the corresponding distributed controller synthesis problem becomes undecidable. The local strategy f for process p is clocked if f can recommend moves for p only based on the *length* of the history of actions that p has executed (as opposed to the local view of p in our work). The com-rigid restriction demands that at any stage of computation, if the local strategy f for process p recommends a set of actions X, then the actions in X involve the same set of processes.

The setting in [48] studies distributed controller synthesis for branching time specifications given as asynchronous transition systems, instead of the usual branching time temporal logics such as CTL, CTL*. The controlled behaviour of the distributed plant is said to satisfy such a specification iff it can be mapped to the unfolding of the specification via a so-called *control-morphism*. It is shown in [48] that this problem is undecidable.

The work [26] investigates distributed controller synthesis for linear time specifications on *finite* behaviours. Note that our work and all the above mentioned work consider infinite behaviours. And in this thesis, we are only interested in infinite behaviours. The main result in [26] states that the distributed controller synthesis problem is decidable provided the specification is a regular (Mazurkiewicz) trace language and the trace alphabet associated with the distributed plant is a *cograph*. More precisely, the latter condition means that there does not exist four distinct actions a_1, a_2, a_3, a_4 of the distributed plant such that a_1 is dependent on a_2, a_2 is dependent on a_3, and a_3 is dependent on a_4. As we will see in section 3.3, it is easy to construct two asynchronous transition systems $\mathcal{A}_1$, $\mathcal{A}_2$, such that $\mathcal{A}_1$ and $\mathcal{A}_2$ have identical trace alphabet that is a cograph, and $\mathcal{A}_1$ is connectedly communicating while $\mathcal{A}_2$ is not connectedly communicating. Moreover, it is also easy to exhibit two asynchronous transition systems $\mathcal{A}_1$, $\mathcal{A}_2$, such that $\mathcal{A}_1$ and $\mathcal{A}_2$ have identical trace alphabet that is *not* a cograph, and $\mathcal{A}_1$ is connectedly communicating while $\mathcal{A}_2$ is not connectedly communicating. Hence the subclass of distributed plant whose trace alphabet satisfies the cograph condition is not comparable with our subclass of CCPs.

The work [25] studies distributed controller synthesis for linear time specifications that concern both finite and infinite behaviour, but imposes a prior a bound on the amount of memory that a local strategy can keep track of. It is shown in [25] that, given a (general) distributed plant and a robust linear time specification in the form of a regular trace-closed subset of $\Sigma^\star \cup \Sigma^\omega$ and a bound m, one can effectively determine whether there exists a winning family of m-memory-bounded local strategies. By an m-memory-bounded local strategy, we mean one whose memory is at most m.

Finally, we note that decentralized controllers have also been studied in the control theory community under the framework of supervisory control of discrete event systems, see for instance [77] and its references. Here one considers the plant to be monolithic but one looks for a set of controllers each of which can control only a subset of the controllable actions.

3.3. The CCP Model

Through the rest of this chapter, we fix a finite set of processes $\mathcal{P}$ and let p, q, range over $\mathcal{P}$. For convenience, we will often write a $\mathcal{P}$-indexed family $\{X_p\}_{p \in \mathcal{P}}$ simply as $\{X_p\}$. A distributed alphabet over $\mathcal{P}$ is a pair (Σ, loc) where Σ is a finite alphabet of actions and $\mathit{loc} : \Sigma \to 2^{\mathcal{P}} \setminus \{\emptyset\}$ identifies for each action, a nonempty set of processes (locations) that take part in each execution of the action. Σ_p is the set of actions that p participates in and it is given by $\{a \in \Sigma \mid p \in \mathit{loc}(a)\}$. Fix such a distributed alphabet through the rest of this chapter.

We will formulate our models of distributed plants in terms of deterministic asynchronous transition systems. We impose determinacy only for convenience. All our results will go through, with minor complications, even in the absence of determinacy. An *asynchronous transition system* (ATS) over (Σ, loc) is a structure $\mathcal{A} = (\{S_p\}, s_{in}, \{\delta_a\}_{a \in \Sigma})$ where S_p is a finite set of p-states for each p and $s_{in} \in \prod_{p \in \mathcal{P}} S_p$. Further, $\delta_a \subseteq \prod_{p \in \mathit{loc}(a)} S_p \times \prod_{p \in \mathit{loc}(a)} S_p$ for each a. The ATS $\mathcal{A}$ is deterministic if for each a, $(s_a, s'_a), (s_a, s''_a) \in \delta_a$ implies $s'_a = s''_a$. From now on we will implicitly assume that the ATSs we

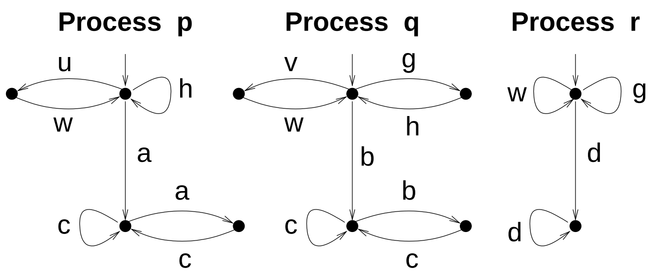


FIGURE 3.1. An asynchronous transition system

encounter are deterministic. Members of $\prod_{p\in\mathcal{P}} S_p$ are referred to as global states. It will be convenient to view the global state s as a map from $\mathcal{P}$ into $\bigcup_{p\in\mathcal{P}} S_p$ such that $s(p) \in S_p$ for every p. For the global state s and $P \subseteq \mathcal{P}$, we will let s_P denote the map s restricted to P. An example of an ATS is shown in figure 3.1, where the locations of an action is assumed are the components in which it appears as a label of a local transition. In particular, we have $loc(u) = \{p\}$, $loc(w) = \{p, q, r\}$.

The dynamics of the ATS $\mathcal{A}$ is given by a transition system $TS_{\mathcal{A}} = (RS_{\mathcal{A}}, s_{in}, \Sigma, \rightarrow_{\mathcal{A}})$ where $RS_{\mathcal{A}} \subseteq \prod_{p\in\mathcal{P}} S_p$, the set of reachable global states, and $\rightarrow_{\mathcal{A}} \subseteq RS_{\mathcal{A}} \times \Sigma \times RS_{\mathcal{A}}$ are least sets satisfying:

- Firstly, $s_{in} \in RS_{\mathcal{A}}$.
- Secondly, suppose $s \in RS_{\mathcal{A}}$ and $s' \in \prod_{p\in\mathcal{P}} S_p$ such that $(s_P, s'_P) \in \delta_a$ and $s_Q = s'_Q$ where $P = loc(a)$ and $Q = \mathcal{P} \setminus P$. Then $s' \in RS_{\mathcal{A}}$ and $s \xrightarrow{a}_{\mathcal{A}} s'$.

We extend $\rightarrow_{\mathcal{A}}$ to sequences in $\Sigma^\star$ in the obvious way. That is, firstly $s \xrightarrow{\varepsilon}_{\mathcal{A}} s$ for every $s \in RS_{\mathcal{A}}$; secondly, if $s \xrightarrow{\sigma}_{\mathcal{A}} s'$ and $s' \xrightarrow{a}_{\mathcal{A}} s''$ where $\sigma \in \Sigma^\star$, $a \in \Sigma$, then $s \xrightarrow{\sigma a}_{\mathcal{A}} s''$. We define $L(\mathcal{A}) = \{\sigma \in \Sigma^\star \mid \exists s.\ s_{in} \xrightarrow{\sigma}_{\mathcal{A}} s\}$.

We shall use (Mazurkiewicz) trace theory ([18]) to capture the notion of connectedly communicating. It will also come in handy in the next section for defining the event structure semantics of asynchronous transition systems. We first recall that a trace alphabet is a pair (Γ, I) where Γ is a finite alphabet set and $I \subseteq \Gamma \times \Gamma$ is an irreflexive and symmetric relation called the *independence relation.* The trace alphabet (Σ, I) induced by the distributed alphabet (Σ, loc) is given by : aIb iff $loc(a) \cap loc(b) = \emptyset$. Clearly I is irreflexive and symmetric. We let $D = (\Sigma \times \Sigma) \setminus I$ denote the dependency relation. In what follows, we let σ, σ' range over $\Sigma^\star$. As usual, the (Mazurkiewicz) trace equivalence relation $\sim_I$ is the least equivalence relation contained in $\Sigma^\star \times \Sigma^\star$ such that $\sigma ab\sigma' \sim_I \sigma ba\sigma'$ whenever $a\, I\, b$. Intuitively, two sequences are trace equivalent iff they differ only in the order of independent letters. In what follows, we will often write $\sim$ instead of $\sim_I$.

We extend the independence relation to $\Sigma^\star$ via: $\sigma\ I\ \sigma'$ iff $a\ I\ b$ for every letter a that appears in σ and every letter b that appears in σ'. We let $\sigma \restriction p$ be the Σ_p-projection of σ. It is the sequence obtained by erasing from σ all appearances of letters that are not in Σ_p. We define $|\sigma|_p = |\sigma \restriction p|$ where $|\tau|$ denotes the length of the sequence τ.

We say that two processes p and q are *separated in* σ if there exist τ, τ' in $\Sigma^\star$ such that $\sigma \sim \tau\tau'$ and $\tau\ I\ \tau'$ and $|\tau|_q = |\tau'|_p = 0$. Thus in the execution represented by σ there can be no flow of information from q to p, or conversely. The ATS $\mathcal{A}$ is *k-communicating* if for every $s \in RS_{\mathcal{A}}$ and every p, q, the following condition is satisfied: Suppose $s \xrightarrow{\sigma}_{\mathcal{A}} s'$ and $|\sigma|_p \geq k$ and $|\sigma|_q = 0$. Then p and q are separated in $\sigma' \in \Sigma^\star$ for any $s' \xrightarrow{\sigma'}_{\mathcal{A}} s''$. We shall say that the ATS $\mathcal{A}$ is *connectedly communicating* iff it is k-communicating for some k.

We note that:

Observation 3.1. *The ATS $\mathcal{A}$ is connectedly communicating iff it is k-communicating for some $k \leq |RS_{\mathcal{A}}|$. Consequently, one can effectively determine whether a given ATS is connectedly communicating.*

Proof. It suffices to show that, if $\mathcal{A}$ is k-communicating, where $k > |RS_{\mathcal{A}}|$, then $\mathcal{A}$ is in fact $|RS_{\mathcal{A}}|$-communicating. Suppose that $\mathcal{A}$ is not $|RS_{\mathcal{A}}|$-communicating. Then there exist $s \xrightarrow{\sigma}_{\mathcal{A}} s' \xrightarrow{\sigma'}_{\mathcal{A}} s''$ and p, q such that

$|\sigma|_p \geq k$, $|\sigma|_q = 0$, but p, q are not separated in σ'. Since $|\sigma| \geq k > |RS_{\mathcal{A}}|$, one can then easily find τ, τ', τ'' in $\Sigma^\star$ such that $\tau' \neq \varepsilon$ and $\sigma = \tau\tau'\tau''$ and $s \xrightarrow{\tau}_{\mathcal{A}} s''' \xrightarrow{\tau'}_{\mathcal{A}} s''' \xrightarrow{\tau''}_{\mathcal{A}} s'$ for some $s''' \in RS_{\mathcal{A}}$. Thus for any $\eta = \tau(\tau')^i\tau''$, it is the case that $s \xrightarrow{\eta}_{\mathcal{A}} s'$. By choosing a sufficiently large i, one arrives at the contradiction that $\mathcal{A}$ is not k-communicating. □

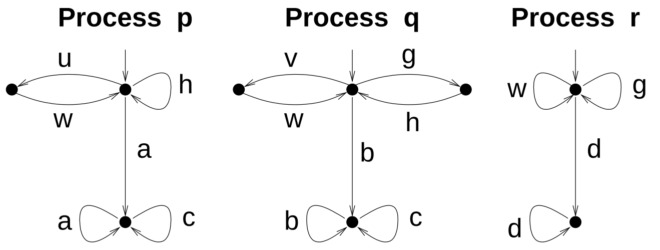


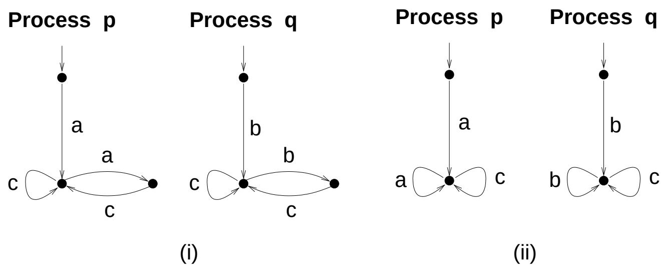
FIGURE 3.2. An ATS which is not connectedly communicating

The ATS shown in figure 3.1 is connectedly communicating, while the ATS shown in figure 3.2 is not. We note that these two ATSs are based on the same distributed alphabet.

Intuitively, let p, q be two processes in an connectedly communicating ATS. Suppose that at a state s, executing a sequence of steps σ leads back to s. Suppose further that, p takes part in σ. However, either q does not take part in σ or more generally, there is no information flow in σ between p and q even though q may also take part in σ. Then in any execution starting from s, there will never be any information flow between p and q. Naturally, connectedly communicating ATSs can model protocols in which components synchronize with each other frequently so that the loss of information on each others' execution remains bounded. Further, if the loss of information of process p on q exceeds the bound, then p will not hear from q anymore, directly or indirectly. This kind of phenomenon commonly occurs in distributed protocols, where one component will only make a bounded number of attempts to establish connections with another component.

We remark that the notion of connectedly communicating is incomparable with the fairness assumption in the study of implementing distributed protocols. Fairness in the the latter context often means simply that every process gets a chance to perform an action infinitely often. Such a fairness condition does not dictate the information flow between different processes. Our study of the connectedly communicating notion is motivated by distributed controller synthesis, in which the information flow between processes plays a crucial role.

Recall from section 3.2 that the work [26] studies distributed controller synthesis where they consider distributed plants based on ATSs whose associated trace alphabet (with the dependence relation) is a cograph. More precisely, (Σ, D) is a cograph iff there does not exists distinct letters a_1, a_2, a_3, a_4 in Σ, such that $a_1 D a_2$ and $a_2 D a_3$ and $a_3 D a_4$. It is easy to see that the trace alphabet induced by the ATS in figure 3.1 (as also that of figure 3.2) is not a cograph. In figure 3.3(i) and 3.3(ii), we show two ATSs over the same distributed alphabet whose induced trace alphabet is a cograph. However, the ATS in figure 3.3(i) is connectedly communicating while that the ATS in figure 3.3(ii) is not. The examples in figures 3.1, 3.2, and 3.3 together show that the subclass of ATSs whose associated trace alphabet is a cograph is incomparable with our subclass of connectedly communicating ATSs.



From now on we will refer to a deterministic connectedly communicating ATS as a CCP.

3.4. The MSO Theory of CCPs

We wish to prove that the MSO theory of the unfolding of every CCP is decidable. To formulate this result we begin with a brief account of event structures.

An event structure (often called a prime event structure) is a triple $ES = (E, \leq, \#)$ where $(E, \leq)$ is a poset such that for every $e \in E$, $\downarrow e = \{e' \in E \mid e' \leq e\}$ is a finite set. And $\# \subseteq E \times E$ is an irreflexive and symmetric relation which satisfies the conflict inheritance axiom: for every $e, e', e'' \in E$, if $e \# e'$ and $e \leq e''$, then $e'' \# e'$. E is the set of events, $\leq$ the causality relation and $\#$ the conflict relation. The minimal causality relation $\lessdot$ is defined as: $e \lessdot e'$ iff $e < e'$ and for every e'', if $e \leq e'' \leq e'$, then $e'' = e$ or $e'' = e'$. The minimal conflict relation $\#_\mu$ is given by: $e \#_\mu e'$ iff $e \# e'$ and $\# \cap (\downarrow e \times \downarrow e') = \{(e, e')\}$. A Σ-labelled event structure is a structure $(E, \leq, \#, \lambda)$ where $(E, \leq, \#)$ is an event structure and $\lambda : E \to \Sigma$ a labelling function.

The non-interleaved branching time behaviour of the ATS $\mathcal{A}$ is naturally given by its event structure unfolding [69]. This Σ-labelled event structure denoted $ES_\mathcal{A}$ is obtained as follows. We first note that $L(\mathcal{A})$ is a trace-closed subset of $\Sigma^\star$ in the sense if $\sigma \in L(\mathcal{A})$ and $\sigma \sim \sigma'$ then $\sigma' \in L(\mathcal{A})$ as well. For a non-null sequence $\sigma \in \Sigma^\star$, we let $last(\sigma)$ denote the last letter appearing in σ. In the present context, we shall view a (Mazurkiewicz) trace as a $\sim$-equivalence class of strings and denote the $\sim$-equivalence class containing the string σ as $[\sigma]_\sim$ and often drop the subscript $\sim$. The partial ordering relation $\sqsubseteq$ over traces is given by : $[\sigma] \sqsubseteq [\sigma']$ iff there exists σ'' in $[\sigma']$ such that σ is a prefix of σ''. A trace $[\sigma]$ is *prime* iff σ is non-null and for every σ' in $[\sigma]$, $last(\sigma) = last(\sigma')$. Thus for a prime trace $[\sigma]$, we can set $last([\sigma]) = last(\sigma)$. Now, $ES_\mathcal{A}$ is defined to be the structure $(E, \leq, \#, \lambda)$ where

- $E = \{[\sigma] \mid \sigma \in L(\mathcal{A})$ and $[\sigma]$ is prime$\}$.
- $\leq$ is $\sqsubseteq$ restricted to $E \times E$.

- $\#$ is given by: $e \# e'$ iff there does *not* exist $\sigma \in L(\mathcal{A})$ such that $e \sqsubseteq [\sigma]$ and $e' \sqsubseteq [\sigma]$, for every $e, e' \in E$.
- $\lambda(e) = \mathit{last}(e)$, for every $e \in E$.

It is easy to check that $ES_{\mathcal{A}}$ is a Σ-labelled event structure. In fact, the labelling function λ will respect the dependency relation D in the sense that if $\lambda(e)\ D\ \lambda(e')$ then it will be the case that $e \leq e'$ or $e' \leq e$ or $e \# e'$ ([18]). And this will endow $ES_{\mathcal{A}}$ with a great deal of additional structure. In particular, it will let us define its MSO theory using just the $\lessdot$ relation and the labelling function as it will turn out below. In what follows, we will often write $ES_{\mathcal{A}}$ as just ES.

In figure 3.3 we display an initial fragment of the event structure unfolding of the ATS shown in figure 3.1. Each event e is represented by a box whose inscription is its label $\lambda(e)$. As usual, directed arrows represent members of the $\lessdot$ relation and the dotted lines[1] represent members of the $\#_{\mu}$ relation. The relations $\leq$ and $\#$ are to be deduced using the transitivity of $\leq$ and the conflict inheritance axiom. In fact, for every event e in $ES_{\mathcal{A}}$, it is easy ([18]) to see that e is the prime trace whose members are the linearizations of the partial order $(\downarrow e, \leq_{\downarrow e})$ subjected to the point-wise application of $\lambda_{\downarrow e}$, where $\leq_{\downarrow e}$, $\lambda_{\downarrow e}$, are the obvious restrictions of $\leq$, λ to $\downarrow e$, respectively. For example, in figure 3.3, the box with inscription v at the bottom leftmost is the event (prime trace) $\{uvwv, vuwv\}$.

We now define the syntax of the MSO logic over $ES_{\mathcal{A}}$ as:

$$\mathit{MSO}(ES_{\mathcal{A}}) ::= R_a(x) \mid x \lessdot y \mid x \in X \mid \exists x\,(\varphi) \mid \exists X(\varphi) \mid \sim \varphi \mid \varphi_1 \vee \varphi_2\ ,$$

where $a \in \Sigma$, $x, y, \ldots$ are individual variables and $X, Y, \ldots$ are set variables. An interpretation $\mathcal{I}$ assigns to every individual variable an event in E and every set variable, a subset of E. The notion of ES satisfying a formula φ

[1]Squiggly edges are commonly used in the literature for this purpose. We have used dotted lines instead to reduce clutter.

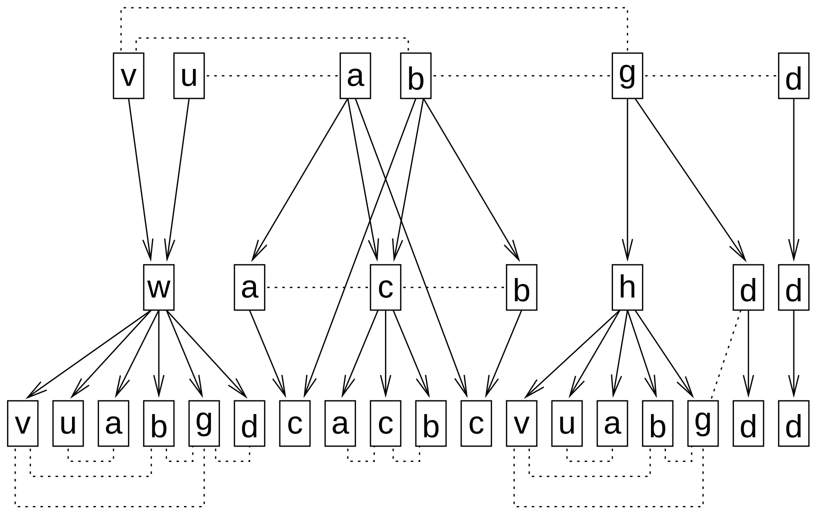


FIGURE 3.3. Event structure

under an interpretation $\mathcal{I}$, denoted $ES \models_{\mathcal{I}} \varphi$, is defined in the obvious way. For example, $ES_{\mathcal{A}} \models_{\mathcal{I}} R_a(x)$ iff $\lambda(\mathcal{I}(x)) = a$; $ES_{\mathcal{A}} \models_{\mathcal{I}} x \lessdot y$ iff $\mathcal{I}(x) \lessdot \mathcal{I}(y)$.

It is a standard observation that $\leq$ can be defined in terms of $\lessdot$ in the presence of set quantification. More precisely, $e \leq e'$ iff

$$\forall X.\big((\forall x. x \in X \rightarrow \forall y.\ y \lessdot x \rightarrow y \in X) \bigwedge (e' \in X)\big) \rightarrow e \in X\ .$$

We next observe that the conflict relation # of $ES_{\mathcal{A}}$ admits an alternative characterization and is thus definable in $MSO(ES_{\mathcal{A}})$.

Observation 3.2. *The conflict relation # is definable in $MSO(ES_{\mathcal{A}})$.*

Proof. Let the relation $\widehat{\#}_D \subseteq E \times E$ be given by: $e \mathrel{\widehat{\#}_D} e'$ iff $e \not\leq e'$ and $e' \not\leq e$ and $\lambda(e)\ D\ \lambda(e')$. Note that Σ is a finite set and hence $\widehat{\#}_D$ is definable in $MSO(ES_{\mathcal{A}})$. Next define $\widehat{\#}$ as: $e \mathrel{\widehat{\#}} e'$ iff there exist $e1, e1' \in E$ such that $e1 \mathrel{\widehat{\#}_D} e1'$ and $e1 \leq e$ and $e1' \leq e'$. We show that $\# = \widehat{\#}$, which at once yields that # is definable in $MSO(ES_{\mathcal{A}})$.

The fact $\# = \widehat{\#}$ follows easily from two basic properties of $ES_{\mathcal{A}}$ ([18]). Firstly, for every $e, e' \in E$, if $e \mathrel{\#_\mu} e'$, then $\lambda(e)\ D\ \lambda(e')$. This immediately yields that $\# \subseteq \widehat{\#}$, due to the conflict inheritance axiom. Secondly, as noted earlier, for every $e, e' \in E$, $\lambda(e)\ D\ \lambda(e')$ implies that $e \leq e'$ or $e' \leq e$ or $e \# e'$. It follows that $\widehat{\#}_D \subseteq \#$ and thus $\widehat{\#} \subseteq \#$. $\square$

The MSO theory of $ES_{\mathcal{A}}$ is the set of sentences (formulae that do not have free occurrences of individual or set variables) given by: $\{\varphi \mid ES_{\mathcal{A}} \models \varphi\}$. The MSO theory of $ES_{\mathcal{A}}$ is said to be decidable if there exists an effective procedure that determines for each sentence φ in $MSO(ES_{\mathcal{A}})$, whether $ES_{\mathcal{A}} \models \varphi$. Finally, by the MSO theory of $\mathcal{A}$ we shall mean the MSO theory of $ES_{\mathcal{A}}$.

It is not difficult to show that the MSO theory of the ATS in figure 3.2 is undecidable.

Proposition 3.3 ([72], attributed to Igor Walukiewicz)**.** *The MSO theory of the event structure of the ATS in figure* 3.2 *is undecidable.*

Proof. Let $ES = (E, \leq, \#, \lambda)$ be the event structure of the asynchronous transition system in figure 3.2. It is easy to see that $X = \{[a^i] \mid i > 0\}$ and $Y = \{[b^j] \mid j > 0\}$ and $Z = \{[a^i b^j c] \mid i, j > 0\}$ are subsets of E. Furthermore, the sets X, Y, Z are definable in $MSO(ES)$. In particular, an event x is in X iff for every $x' \leq x$, $\lambda(x') = a$, an event z is in Z iff $\lambda(z) = c$, and for every $z' < z$, $\lambda(z')$ is a or b, and there exists $z1, z2$ such that $z1 < z, \lambda(z1) = a$ and $z2 < z, \lambda(z2) = b$.

We view Z as an encoding of the 2-dimensional grid $\mathbb{N} \times \mathbb{N}$ with the grid point (i, j) being represented by the event $[a^{i+1} b^{j+1} c]$. In $MSO(ES)$, we can construct a formula *right-succ*(u, v) with free variables u, v which asserts that u, v are in Z and v is the *right-successor* of u, that is, if u denotes (i, j) then v denotes $(i + 1, j)$. It is easy to see that if u, v are in Z, then v is the right-successor of u iff there exist x, x' and y such that $x, x' \in X$, $y \in Y$, $x \lessdot u$, $y \lessdot u$, $x' \lessdot v$, $y \lessdot v$ and $x \lessdot x'$. Similarly, in $MSO(ES)$, we can construct a formula *up-succ*(u, v) with free variables u, v which asserts that u, v are in

Z and v is the *up-successor* of u, that is, if u denotes (i, j) then v denotes $(i, j + 1)$.

As a result, the following grid coloring problem which is known to be undecidable [46] can be reduced to the decision problem of $MSO(ES)$. An instance of the coloring problem consists of a finite set of colors $Col = \{c_0, c_1, \ldots, c_n\}$ and two functions $R : Col \to 2^{Col}$ and $U : Col \to 2^{Col}$. The problem is to determine if there exists a coloring function $f : \mathbb{N} \times \mathbb{N} \to Col$ such that $f(0, 0) = c_0$ and for each (i, j) it is the case that $f(i+1, j) \in R(i, j)$ and $f(i, j + 1) \in U(i, j)$. $\square$

Our main logical result is:

Theorem 3.4. *The MSO theory of every CCP is decidable.*

We shall use this logical result in the next section to prove the decidability of the distributed controller synthesis problem where the distributed plant is based on a CCP and the specification is robust.

In what follows, we establish theorem 3.4. Let $\mathcal{A}$ be a k-communicating ATS where $k \leq |RS_{\mathcal{A}}|$. Let $TR = (\Sigma^\star, \{succ_a\}_{a \in \Sigma})$ be the full Σ-tree ([73]), where $succ_a = \{(\sigma, \sigma a) \mid \sigma \in \Sigma^\star\}$ for $a \in \Sigma$. Members of $\Sigma^\star$ are nodes with ε being the root. We shall denote the standard MSO logic of n-successors ($|\Sigma| = n$) interpreted over TR as $MSO(TR)$. We refer to section 2.2 for the definition of $MSO(TR)$.

We shall show that the structure $(E, \lessdot, \lambda)$ can be embedded in TR and that this embedding can be defined in $MSO(TR)$. This will at once yield theorem 3.4 by Rabin's famous result that $MSO(TR)$ is decidable [63].

In what follows, we fix a total order *lex* on Σ. Often, we refer to this order implicitly, for example, by speaking of a being less than b. Clearly *lex* induces a total order over $\Sigma^\star$ which we shall refer to as the lexicographic order. For an event e in E with $e = [\sigma]$, we let $lin(e)$ be the lexicographically least member in $[\sigma]$. Set $LEX_{\mathcal{A}} = \{lin(e) \mid e \in E\}$. In what follows, we will write $LEX_{\mathcal{A}}$ as just LEX. Clearly $LEX \subseteq \Sigma^\star$ and hence members of LEX can be looked upon as distinguished nodes in the tree TR. A pleasant fact is that LEX is definable in $MSO(TR)$.

Lemma 3.5. *One can effectively construct a formula $\varphi_{LEX}(x)$ with one free individual variable x such that for any interpretation $\mathcal{I}$ of TR, $TR \models_{\mathcal{I}} \varphi_{LEX}(x)$ iff $\mathcal{I}(x) \in LEX$.*

Proof. It is easy to show that $L_{events} = \{\sigma \mid [\sigma] \in E\}$ is a regular trace-closed subset of $\Sigma^\star$ and is hence a regular trace language ([18]). It is known that the collection $\widehat{L}_{lex}$ obtained by picking the lexicographically least member of each $\sim$-equivalence class of a regular trace language $\widehat{L}$ is, in turn, a regular language [18]. Thus LEX is a regular subset of $\Sigma^\star$ and we can effectively construct from $\mathcal{A}$, a deterministic finite state automaton accepting LEX. Further, one can describe the successful runs of this automaton in the form of a formula $\varphi_{LEX}(x)$ ([73]). $\square$

Define now the relation $\lessdot_{LEX} \subseteq LEX \times LEX$ by: $\sigma \lessdot_{LEX} \sigma'$ iff $[\sigma] \lessdot [\sigma']$ in $ES_{\mathcal{A}}$. Define also the map λ_{LEX} as $\lambda_{LEX}(\sigma) = last(\sigma)$ for every $\sigma \in LEX$. It now follows that $(LEX, \lessdot_{LEX}, \lambda_{LEX})$ is isomorphic to the structure $(E, \lessdot, \lambda)$. Hence if we show that $\lessdot_{LEX}$ is definable in $MSO(TR)$ then we are done. In this light, the following result is crucial.

Lemma 3.6. *There exists a constant K (which can be effectively computed from $\mathcal{A}$) with the following property: Suppose $w = a_1 \ldots a_m, w' = b_1 \ldots b_n \in LEX$. Suppose further, $w \lessdot_{LEX} w'$ and w is not a prefix of w'. Then $|a_i a_{i+1} \ldots a_m| \leq K$, where i is the least index such that $a_i \neq b_i$.*

Proof. Let $e = [w]$ and $e' = [w']$ so that $e \lessdot e'$. It follows from the definition of ES that $w' \sim w\tau$ for some τ in Σ^+. Hence b_i is less than a_i. We first show that $b_i \; I \; a_i a_{i+1} \ldots a_m$. Suppose $b_i \; I \; a_i a_{i+1} \ldots a_m$ does not hold. Let j $(i \leq j \leq m)$ be the least index such that $a_j \; D \; b_i$. A basic property of traces

([18]) is that if $a\,D\,b$ then the $\{a,b\}$-projection of $\sigma 1$ is identical to the $\{a,b\}$-projection of $\sigma 2$ whenever $\sigma 1 \sim \sigma 2$. It follows that $a_j = b_i$. But then b_i being less than a_i would imply that $\widehat{w} = a_1 \ldots a_{i-1} b_i a_i \ldots a_{j-1} a_{j+1} \ldots a_m \sim w$ and clearly $\widehat{w}$ is lexicographically less than w, a contradiction.

Having established that $b_i \;\; I \;\; a_i a_{i+1} \ldots a_m$, we next show that $|a_i a_{i+1} \ldots a_m| \leq k|\mathcal{P}|$. Suppose $|a_i a_{i+1} \ldots a_m| > k|\mathcal{P}|$. Then there exists p such that $|a_i a_{i+1} \ldots a_m|_p \geq k+1$. Pick l $(i \leq l \leq m)$ such that $p \in loc(a_l)$ and $|a_i a_{i+1} \ldots a_{l-1}|_p \geq k$. Let $q \in loc(b_i)$. Since $b_i \; I \; a_i a_{i+1} \ldots a_m$, we have $|a_i a_{i+1} \ldots a_{l-1}|_q = 0$. It follows that p, q must be separated in $a_l a_{l+1} \ldots a_m b_i \tau'$, where $w' \sim w b_i \tau'$. This contradicts the facts that $[w']$ (hence $[a_l a_{l+1} \ldots a_m b_i \tau']$) is a prime trace and $p \in loc(a_l)$ and $q \in loc(b_i)$. $\square$

We can now use lemma 3.6 to show that $\lessdot_{LEX}$ is expressible in $MSO(TR)$.

Lemma 3.7. *One can effectively construct a formula $\varphi_{\lessdot}(x,y)$ in $MSO(TR)$ with two free individual variables x and y such that, for any interpretation $\mathcal{I}$ of TR, $TR \models_{\mathcal{I}} \varphi_{\lessdot}(x,y)$ iff $\mathcal{I}(x), \mathcal{I}(y) \in LEX$ and $\mathcal{I}(x) \lessdot_{LEX} \mathcal{I}(y)$.*

Proof. Let $w, w' \in LEX$. Consider the condition $C1$ given by:

$$C1\text{:}\quad \begin{array}{l} w \text{ is a proper prefix of } w' \text{ and } last(w)\ D\ last(w') \\ \text{and } last(w)\ I\ w'' \text{ where } w' = ww''. \end{array}$$

It is easy to see that if $C1$ is satisfied then $w \lessdot_{LEX} w'$ and moreover, $C1$ is definable in $MSO(TR)$. Let K be the constant established in lemma 3.6. Now consider the following conditions:

$$\begin{array}{rl} C2.1: & w = w_0 a_1 a_2 \ldots a_l \text{ with } l \leq K \text{ and} \\ & w' = w_0 w'_1 a_1 w'_2 a_2 \ldots w'_l a_l w'_{l+1} last(w'). \\ C2.2: & w'_i\ I\ a_j \text{ for } 1 \leq i \leq j \leq l \text{ and } a_l\ I\ w'_{l+1}. \\ C2.3: & a_l\ D\ last(w'). \end{array}$$

Let $C2$ be the conjunction of $C2.1$, $C2.2$ and $C2.3$. It is easy to see that if $C2$ is satisfied then $w \lessdot_{LEX} w'$ and also that $C2$ is definable in $MSO(TR)$.

Next we show that if $w \lessdot_{LEX} w'$ then $C1$ or $C2$ is satisfied, which will then complete the proof.

Suppose $w \lessdot_{LEX} w'$. If w is a prefix of w', then it is clear that $C1$ is satisfied. Now suppose w is not a prefix of w'. Let $w = w_0 a_1 a_2 \ldots a_l$ and $w' = w_0 b_1 b_2 \ldots b_m$ where w_0 is the longest common prefix of w, w'. It follows from lemma 3.6 that $l \leq K$. Since $w \lessdot_{LEX} w'$, there exists $\tau \in \Sigma^\star$ such that $w' \sim w\tau \mathit{last}(w')$ and $\mathit{last}(w)\ I\ \tau$, $\mathit{last}(w)\ D\ \mathit{last}(w')$. Thus w' takes the form $w_0 w'_1 b_{h_1} w'_2 b_{h_2} \ldots w'_l b_{h_l} w'_{l+1} b_m$ where

- $w'_i\ I\ b_{h_j}$ for $1 \leq i \leq j \leq l$, and $b_{h_l}\ I\ w'_{l+1}$.
- $b_{h_l}\ D\ b_m$.
- $b_{h_1} b_{h_2} \ldots b_{h_l} \sim a_1 a_2 \ldots a_l$.

We show that in fact $b_{h_1} = a_1, b_{h_2} = a_2, \ldots, b_{h_l} = a_l$.

Suppose $b_{h_1} = a_1$ does not hold. Let j $(1 < j \leq l)$ be the least index such that $b_{h_j} = a_1$. It follows that $b_{h_j}\ I\ b_{h_1} b_{h_2} \ldots b_{h_{j-1}}$ and thus $\widehat{w} \sim w'$ where $\widehat{w} = w_0 w'_1 b_{h_j} b_{h_1} w'_2 b_{h_2} \ldots b_{h_{j-1}} w'_j w'_{j+1} b_{h_{j+1}} \ldots b_{h_l} w'_{l+1} b_m$. Since w is lexicographically less than $w_0 b_{h_1} b_{h_2} \ldots b_{h_l}$, $b_{h_j} = a_1$ is less than b_{h_1} and thus $\widehat{w}$ is lexicographically less than w', a contradiction. Inductively, one can show that $b_{h_2} = a_2, \ldots, b_{h_l} = a_l$. □

We can now establish theorem 3.4.

Proof of theorem 3.4. Define the map $\|\cdot\|$ from $\mathit{MSO}(ES_{\mathcal{A}})$ into $\mathit{MSO}(TR)$ inductively:

- Firstly, $\|R_a(x)\| = \exists y\, \mathit{succ}_a(y, x)$ and $\|x \lessdot y\| = \varphi_{\lessdot}(x, y)$ where $\varphi_{\lessdot}(x, y)$ is the formula established in lemma 3.7.
- Secondly, we define $\|x \in X\| = x \in X$. Further, $\|\exists x\, (\Psi)\| = \exists x\, (\varphi_{LEX}(x) \wedge \|\Psi\|)$ and $\|\exists X\ (\Psi)\| = \exists X\ ((\forall x \in X\ \varphi_{LEX}(x)) \wedge \|\Psi\|)$ where $\varphi_{LEX}(x)$ is the formula established in lemma 3.5.
- Finally, $\|{\sim}\ \Psi\| = {\sim}\|\Psi\|$ and $\|\Psi1 \vee \Psi2\| = \|\Psi1\| \vee \|\Psi2\|$.

One verifies that for each sentence Ψ in $MSO(ES_{\mathcal{A}})$, $ES_{\mathcal{A}} \models \Psi$ iff $TR \models \|\Psi\|$, by a routine induction on the structure of Ψ. This then shows that $MSO(ES_{\mathcal{A}})$ is decidable, following Rabin's famous result that $MSO(TR)$ is decidable [63]. $\square$

As for complexity, we have:

Theorem 3.8. *Given a CCP $\mathcal{A}$ and a sentence Ψ in $MSO(ES_{\mathcal{A}})$, the complexity of the above decision procedure for determining whether $ES_{\mathcal{A}} \models \Psi$ is $tower(O(|RS_{\mathcal{A}}| \cdot quan(\Psi)),\ O(|\Psi|))$ where $|\Psi|$ is the* size *of Ψ and $quan(\Psi)$ is the total number of quantifiers in Ψ. And $tower(a,n)$ is inductively given by: $tower(a,1) = a$ and $tower(a,n+1) = a^{tower(a,n)}$.*

In particular, for a fixed $\mathcal{A}$, then the complexity of testing whether $ES_{\mathcal{A}} \models \Psi$ is non-elementary in $|\Psi|$.

In what follows, we prove theorem 3.8. We fix a sentence Ψ. First, we make clear the notion of the *size* of Ψ. It is a standard observation (see [73]) that Ψ can be put into the prenex normal form as

$$\Psi' = B_1 B_2 \ldots B_m B'_1 B'_2 \ldots B'_{m'}\ \psi\ ,$$

such that $ES_{\mathcal{A}} \models \Psi$ iff $ES_{\mathcal{A}} \models \Psi'$. And ψ is a quantifier free formula; each B_i, $i = 1, 2, \ldots, m$, is either an existential block of set quantification $\exists X_{i1} \exists X_{i2} \ldots \exists X_{ig_i}$, or a universal block of set quantification $\forall X_{i1} \forall X_{i2} \ldots \forall X_{ig_i}$; and if B_i is existential (respectively universal), then B_{i+1} is universal (respectively, existential). Moreover, B'_i, $i = 1, 2, \ldots, m'$ are blocks of individual quantification with analogous forms as B_i, $i = 1, 2, \ldots, m$. The size of Ψ is then defined to be $m + m'$.

The complexity stated in theorem 3.8 may seem extremely high at first sight. However, we emphasize that, for the purpose of model checking and controller synthesis of practical distributed protocols, the size of MSO formulae over $ES_{\mathcal{A}}$ is usually very small.

Proof of theorem 3.8. With the above notations, it follows from the definition of $\|\cdot\|$ that $\|\Psi'\|$ will have the form

$$C_1 C_2 \dots C_m C'_1 C'_2 \dots C'_{m'} \ \ \|\psi\|$$

where each C_i is of the form

$$\exists X_{i1}.\ (\forall x.\ x \in X,\ \varphi_{LEX}(x))\ \wedge \dots \exists X_{ig_i}(\forall x.\ x \in X,\ \varphi_{LEX}(x)) \wedge$$

or

$$\forall X_{i1}.\ (\forall x.\ x \in X,\ \varphi_{LEX}(x))\ \wedge \dots \forall X_{ig_i}.\ (\forall x.\ x \in X, \varphi_{LEX}(x))\ \wedge .$$

Similarly, C'_i is of the form

$$\exists x_{i1}.\ \varphi_{LEX}(x_{i1})\ \wedge \dots \exists x_{ih_i}.\ \varphi_{LEX}(x_{ig_i}) \wedge$$

or

$$\forall x_{i1}.\ \varphi_{LEX}(x_{i1})\ \wedge \dots \forall x_{ih_i}.\ \varphi_{LEX}(x_{h_i})\ \wedge\ \ .$$

To test the truth of $\|\Psi'\|$ in $MSO(TR)$, the decision procedure (see [73] for details) builds inductively non-deterministic Rabin tree automata for the subformulas

$$\begin{aligned}
&\|\psi\|\,,\\
&C'_{m'}\ \ \|\psi\|\,,\\
&C'_{m'-1}C'_{m'}\ \ \|\psi\|\,,\\
&\dots,\\
&C'_1 C'_2 \dots C'_{m'}\ \ \|\psi\|\,,\\
&C_m C'_1 C'_2 \dots C'_{m'}\ \ \|\psi\|\,,\\
&C_{m-1} C_m C'_1 C'_2 \dots C'_{m'}\ \ \|\psi\|\,,\\
&\dots,\\
&C_1 \dots C_{m-1} C_m C'_1 C'_2 \dots C'_{m'}\ \ \|\psi\|\,.
\end{aligned}$$

Each quantifier alternation incurs an exponential blow up on the number of states of the tree automaton. Since the formula $\varphi_{LEX}(x)$ checks whether x is in LEX, which is a subset of $L(\mathcal{A})$, it will have $O(|RS_{\mathcal{A}}|)$ quantifier-free subformulas. Hence each C_i will have $O(|RS_{\mathcal{A}}| \cdot quan(\Psi))$ quantifier-free subformulas. To sum up, it follows that the complexity of determining the truth of $\|\Psi'\|$ in $MSO(TR)$ is $tower(O(|RS_{\mathcal{A}}| \cdot quan(\Psi')),\ O(|\Psi'|))$. This completes the proof, as $quan(\Psi') = quan(\Psi)$ and $|\Psi'| = |\Psi|$. $\square$

3.5. The CCP Plant Model

The goal of this section is to formulate the distributed plant model based on CCPs and the associated local strategies. We also define formally the notion of robust linear time specifications.

A *distributed plant* is a structure $\mathcal{A} = (\{S_p^{env}\}, \{S_p^{sys}\}, s_{in}, \Sigma^{env}, \Sigma^{sys}, \{\delta_a\}_{a \in \Sigma})$ where $(\{S_p\}, s_{in}, \{\delta_a\}_{a \in \Sigma})$ is a deterministic ATS over (Σ, loc), called the underlying ATS of $\mathcal{A}$ with $S_p = S_p^{env} \cup S_p^{sys}$ and $S_p^{env} \cap S_p^{sys} = \emptyset$ for each p. Further, $\{\Sigma^{env}, \Sigma^{sys}\}$ is a partition of Σ such that for each a in Σ^{env}, $|loc(a)| = 1$. Finally, suppose $(s_a, s'_a) \in \delta_a$ and $p \in loc(a)$. Then $s_a(p) \in S_p^{env}$ iff $a \in \Sigma^{env}$ and hence $loc(a) = \{p\}$.

The sets S_p^{env}, S_p^{sys} are respectively the p-environment and p-system states. The sets Σ^{env} and Σ^{sys} are the environment (uncontrollable) and system (controllable) actions respectively. Each component interacts with its local environment and these interactions are enabled only when the component is in one of its environment states. We note that although the underlying ATS is deterministic, in general, a menu of controllable actions involving different processes will be available for the controller at each stage as the distributed plant evolves. This will be the case even for the local strategies we define below. Through the rest of the section, we fix a distributed plant $\mathcal{A}$ as above. When talking about the behavioural aspects of $\mathcal{A}$, we shall identify it with its underlying ATS and will often drop the subscript $\mathcal{A}$. We will also say the distributed plant is a CCP in case its underlying ATS is.

Figure 3.4 shows an example of a distributed plant. There are four processes p, q, `B1` and `B2`. The set of environment actions is { `pProduce`, `pConsume`, `qProduce`, `qConsume` }. All other actions are system actions.

Intuitively, the distributed plant in figure 3.4 models that processes p, q communicate through two buffers `B1`, `B2`. Process p behaves as follows. Initially, the local environment of process p decides whether it wants to produce (`pProduce`) or consume (`pConsume`) data. Next process p can make a choice among { `pUseB1`, `pUseB2`, `pUseB1Only`, `pUseB2Only` }, where `pUseB1` means that `B1` should be used for the current produce or consume request (and thus `B2` should be used for the successive produce or consume request), while `pUseB1Only` indicates that `B1` should be used for all produce or consume requests from now on. The actions `pUseB2`, `pUseB2Only` have analogous meanings. The actions `pReadB1`, `pWriteB1` (respectively, `pReadB2`, `pWriteB2`) model that p reads or writes to `B1`. The behaviours of processes q, `B1` and `B2` are now clear from figure 3.4. It is easy to verify that the distributed plant shown in figure 3.4 is a CCP.

In what follows, we define the distributed controller synthesis problem formally. Recall the set $L(\mathcal{A})$ defined in section 3.3. Members of $L(\mathcal{A})$ are referred to as *plays*. The set of infinite plays $L^\omega(\mathcal{A})$ is defined in the obvious way. That is, $\sigma \in \Sigma^\omega$ is in $L^\omega(\mathcal{A})$ iff every finite prefix of σ is in $L(\mathcal{A})$. We are interested in distributed strategies obtained by piecing together local strategies and the local views of a play will be instrumental in determining local strategies.

Let $\sigma = a_1 \ldots a_n$ be a play in $L(\mathcal{A})$. The *p-view* of σ denoted $\downarrow^p(\sigma)$ is the subsequence $a_{h_1} \ldots a_{h_m}$ such that $H = \{h_1, h_2, \ldots, h_m\}$ is the least subset of $\{1, 2, \ldots, n\}$ which satisfies:

- Firstly, h_m is the largest index in $\{1, 2, \ldots, n\}$ such that p is in $loc(a_{h_m})$.
- Secondly, if $i \in H$ and $j < i$ and $a_j \; D \; a_i$, then $j \in H$.

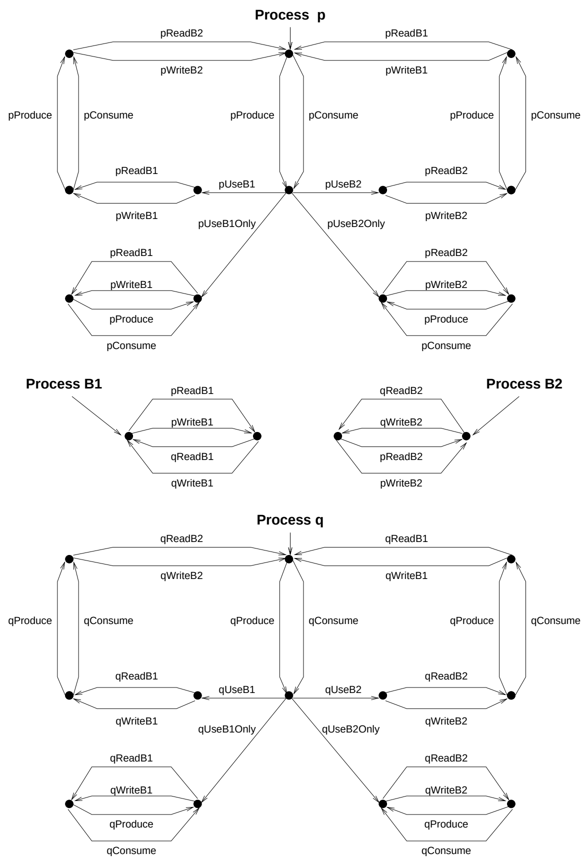


FIGURE 3.4. A CCP distributed plant

In other words, $\downarrow^p (\sigma)$ is the maximum amount of the current play that p knows about where this knowledge is gathered by its participation in the actions that have occurred in the play *and* the information it acquires as a result of synchronizations with other agents, directly or indirectly. It is easy to verify that for any play σ in $L(\mathcal{A})$ and any p, the p-view of σ is also in $L(\mathcal{A})$. For example, in figure 3.4, the p-view of the play `pConsume pUseB1 qProduce qUseB1` is `pConsume pUseB1`, while the p-view of the play

$\sigma =$ `pConsume pUseB1 qProduce qUseB1 qWriteB1 pReadB1`

is σ itself.

It will be convenient to define the set of actions that can potentially occur at a local state. For $u \in S_p$ we let $act(u)$ be the set given by: $a \in \Sigma_p$

is in $act(u)$ iff there exists (s_a, s'_a) in δ_a with $s_a(p) = u$. A *p-strategy* is a function $f : L(\mathcal{A}) \rightarrow 2^{\Sigma_p}$ which satisfies: Suppose $\sigma \in L(\mathcal{A})$ and $s_{in} \xrightarrow{\sigma} s$ with $s(p) = u$. Then $f(\sigma) \subseteq act(u)$ and moreover $f(\sigma) = act(u)$ in case $u \in S_p^{env}$. Thus a p-strategy recommends a subset of the structurally possible Σ_p-actions at the current p-state. It does so without restricting in any way the environment's choices.

The p-strategy f is said to be *local* if it satisfies: for every $\sigma, \sigma' \in L(\mathcal{A})$, $\downarrow^p(\sigma) \sim \downarrow^p(\sigma')$ implies $f(\sigma) = f(\sigma')$. Hence a local p-strategy depends only on the (partially ordered!) p-view of the play.

We now define a *distributed strategy* $Str = \{Str_p\}$ to be a family of local p-strategies, one for every p.

Let $Str = \{Str_p\}$ be a distributed strategy. The set of plays according to Str denoted $L(Str)$ is defined inductively by: Firstly, $\varepsilon \in L(Str)$. Secondly, if $\sigma \in L(Str)$ and $\sigma a \in L(\mathcal{A})$ such that $a \in Str_p(\sigma)$ for every $p \in loc(a)$, then $\sigma a \in L(Str)$. That is, an action a is allowed to execute only when it is recommended by every process taking part in a. In what follows, we will assume without loss of generality that $TS_{\mathcal{A}}$ has no deadlocks; more precisely, every reachable global state has a successor state reachable via a transition. Thus if a play according to a strategy cannot be extended it is only due to the local strategies not being able to agree on executing any system action. We will say that a strategy Str is *non-blocking* in case every play in $L(Str)$ can be extended to a longer play in $L(Str)$. This notion does not rule out the possibility of a play being extended indefinitely by just the execution of environmental actions. However one can rule out such plays by choosing the specification suitably.

To define linear time specifications, we first define the set of infinite plays according to the strategy Str denoted $L^\omega(Str)$ in the obvious way. That is, $\sigma \in \Sigma^\omega$ is in $L^\omega(Str)$ iff every finite prefix of σ is in $L(Str)$. A linear time *specification* is an ω-regular subset of Σ^ω which is assumed to be presented in a finite way, say, as a Büchi automaton. Unless stated otherwise, by a specification, we shall mean a linear time specification. For example, for the distributed plant in figure 3.4, one specification could demand that data transmission between p and q always succeeds whenever p wants to produce data and q wants to consume data (or q wants to produce data and p wants to consume data).

Let L_{spec} be a specification. A distributed strategy Str is winning for L_{spec} iff Str is non-blocking and $L^{\omega}(Str) \subseteq L_{spec}$. A winning distributed strategy for L_{spec} is called a *distributed controller* for the pair $(\mathcal{A}, L_{spec})$. The distributed controller synthesis problem we wish to solve is: given a pair $(\mathcal{A}, L_{spec})$ where $\mathcal{A}$ is a CCP, determine whether there exists a distributed controller for L_{spec}. We will be mainly interested in showing here that this problem is effectively solvable if the specification L_{spec} is *robust*.

To pin down robustness, we extend $\sim$ to Σ^{ω}. This can be done in a number of equivalent ways. For our purposes it will do to define it as follows: Suppose $\sigma, \sigma' \in \Sigma^{\omega}$. Then $\sigma \sim \sigma'$ iff $\sigma \restriction p = \sigma' \restriction p$ for every p. We say that the specification L_{spec} is *robust* iff for every $\sigma, \sigma' \in \Sigma^{\omega}$, if $\sigma \in L_{spec}$ and $\sigma \sim \sigma'$, then $\sigma' \in L_{spec}$. In other words, the equivalence classes of $\sim$ over Σ^{ω} are infinite traces over our trace alphabet (Σ, I) ([18]). And a robust specification is an ω-regular trace language ([18]). In fact, in the distributed setting, since executions are naturally partially ordered, it would be natural to consider robust specifications.

3.6. Decidability Results

In this section, we establish decidability results for distributed controller synthesis associated with CCP-based distributed plants for both robust linear time and branching time specifications.

3.6.1. Robust Linear Time Specifications.

We begin with the case for robust linear time specifications.

Theorem 3.9.

(i) *Given a CCP distributed plant $\mathcal{A}$ and a robust specification L_{spec}, we can effectively determine whether there exists a distributed controller for $(\mathcal{A}, L_{spec})$.*

(ii) *Further, if such a distributed controller exists, then we can effectively synthesize a* finite state *one presented in the form of a CCP.*

In what follows, we prove theorem 3.9(i). The proof of theorem 3.9(ii) is technically of very different nature from that of theorem 3.9(i) and deserves an independent treatment. Hence we defer the proof of theorem 3.9(ii) to section 3.7.

We shall assume $\mathcal{A}$ is a CCP and L_{spec} is robust. We shall show that the existence of a distributed controller for $(\mathcal{A}, L_{spec})$ can be asserted as a sentence in $MSO(ES_{\mathcal{A}})$. Theorem 3.9(i) will then follow at once from theorem 3.4.

In what follows, we let $ES_{\mathcal{A}} = (E, \leq, \#, \lambda)$ and often write ES instead of $ES_{\mathcal{A}}$. A *configuration* of ES is a subset $c \subseteq E$ such that $\downarrow c = c$ (where $\downarrow c = \cup_{e \in c}(\downarrow e)$) and $(c \times c) \cap \# = \emptyset$. Let c be a finite configuration. Then it is well-known ([18]) that the Σ-labelled poset $(c, \leq_c, \lambda_c)$ where $\leq_c$ and λ_c are the obvious restrictions, represents a trace in the following sense. The set of linearizations of $(c, \leq_c)$ (subjected to the point-wise application of λ_c) will be a trace, viewed as a $\sim$-equivalence class of strings. In fact finite and infinite configurations on the one hand and finite and infinite traces on the other hand, represent each other.

Firstly we show that in $MSO(ES)$ one can construct a formula $\mathit{infinite}(X)$ with one free set variable X which asserts that X is an infinite set of events. Consequently, in $\mathit{MSO}(\mathit{ES})$ one can define a formula $\mathit{fin\text{-}conf}(X)$ ($\mathit{inf\text{-}conf}(X)$) asserting that X is a finite (infinite) configuration.

Claim 3.10. *Let $\mathcal{A}$ be a CCP. Then in $\mathit{MSO}(\mathit{ES}_{\mathcal{A}})$ one can construct a formula $\mathit{infinite}(X)$ with one free set variable X such that for any interpretation $\mathcal{I}$ of $\mathit{ES}_{\mathcal{A}}$, $\mathit{ES}_{\mathcal{A}} \models_{\mathcal{I}} \mathit{infinite}(X)$ iff $\mathcal{I}(X)$ is an infinite set of events.*

Proof. For each event e in $\mathit{ES}_{\mathcal{A}}$, we define the *depth* of e, denoted $\mathit{depth}(e)$, to be the largest m such that there exists $e_1 \lessdot e_2 \lessdot \cdots \lessdot e_m = e$. It is easy to see that every event has a finite depth.

We observe the following:

- The predicate $\mathit{dc}(X, Y)$ is definable in $\mathit{MSO}(\mathit{ES}_{\mathcal{A}})$ where $\mathit{dc}(X, Y)$ stands for Y is the downward-closure of X, that is, $Y = \downarrow X$. One has to simply assert that for every y, it is the case that $y \in Y$ iff there exists $x \in X$ such that $y \leq x$.
- The predicate $\mathit{infchain}(X)$ is definable in $\mathit{MSO}(\mathit{ES}_{\mathcal{A}})$, where $\mathit{infchain}(X)$ asserts that X is non-empty and for every x in X there exists y in X such that $x < y$. It is a standard fact that that a set being non-empty can be asserted in the MSO logic and hence $\mathit{infchain}(X)$ says that every element of X lies on an infinite chain and this in turn implies that X contains an infinite chain.
- X is infinite iff there exist Y and Z such that Y is the downward-closure of X and $Z \subseteq Y$ and $\mathit{infchain}(Z)$.

The last observation clearly shows that $\mathit{infinite}(X)$ is definable in $\mathit{MSO}(\mathit{ES}_{\mathcal{A}})$. To prove that this assertion holds we first note that a set of events X is infinite iff $Y = \downarrow X$ is infinite. This follows from the fact that for every event e, $\downarrow e$ is finite. So assume that Y is a downward-closed set of events. We claim that Y is an infinite set iff it contains a set Z such that $\mathit{infchain}(Z)$ holds. The "if" part is clear and so assume that Y is an infinite set. We now consider the tree T whose nodes are given inductively by: $\top$ is a special element and it is the root of the tree and we define $\mathit{last}(\top) = \top$. Inductively, suppose ρ is a node and $\mathit{last}(\rho) = u$ and $u \ll e$ with $e \in Y$ and $\ll$ defined as below. Then ρe is a node and $\mathit{last}(\rho e) = e$. The relation $\ll$ is given by: $u \ll e$ iff one of the following holds:

- $u = \top$, $e \in Y$, and $\mathit{depth}(e) = 1$.
- $u, e \in Y$, $u \lessdot e$, and $\mathit{depth}(e) = \mathit{depth}(u) + 1$.

We claim that T is finitely branching. This follows from the fact that for every depth m, there are only finitely many events of depth m. This in turn follows easily by induction on m because, at every global state, only a bounded number of transitions are enabled.

Now we can apply König's lemma ([37]) to deduce that the tree T contains an infinite path. It follows that Y contains a set Z with the property $\mathit{infchain}(Z)$. $\square$

Next we define, for $E' \subseteq E$, the p-view of E' denoted $p\text{-}\mathit{view}(E')$ to be the set of events given by: $e' \in p\text{-}\mathit{view}(E')$ iff there exists $e'' \in E'$ such that $e' \leq e''$ and $p \in \mathit{loc}(\lambda(e''))$. Again it is easy to see that we can define a formula $p\text{-}\mathit{view}(X, Y)$ asserting that Y is the p-view of X. Let σ be a play of $\mathcal{A}$. By the observation above, there is a unique configuration c of $ES_{\mathcal{A}}$ which represents $[\sigma]$ in the sense that $[\sigma]$ is the the set of linearizations of the Σ-labelled poset $(c, \leq_c, \lambda_c)$. Let $\downarrow^p(\sigma) = \tau$. Then it is easy to see that $p\text{-}\mathit{view}(c)$ is the configuration which represents $[\tau]$.

Now let Str be a distributed strategy. From the definitions, it follows that $L(\mathit{Str})$ is trace-closed. Hence for each $\sigma \in L(\mathit{Str})$ we will have that $[\sigma] \subseteq L(\mathit{Str})$ and moreover, by the observation above, there will be a unique finite configuration in ES that corresponds to $[\sigma]$. We will say that E_{Str} is the *set of Str-events* and define it to be the set given by: $e \in E$ is in E_{Str} iff there exists $\sigma \in L(\mathit{Str})$ such that $e = [\sigma]$. We will say that E' is *good* in case there exists a distributed strategy Str such that E' is the set of Str-events. We can construct a formula $\mathit{Good}(X)$ which will assert that

X is good. For arguing this, it will be convenient to assume the transition relation $\Rightarrow \subseteq C_{fin} \times E \times C_{fin}$ where C_{fin} is the set of finite configurations of *ES* and $\Rightarrow$ is given by: $c \overset{e}{\Rightarrow} c'$ iff $e \notin c$ and $c' = c \cup \{e\}$. The formula *Good*(X) will be a conjunction of the following properties all of which are easily definable in *MSO*(*ES*).

- X is a nonempty set and for every finite configuration Y contained in X, if $Y \overset{e}{\Rightarrow} Y'$ and $\lambda(e) \in \Sigma^{env}$ then $Y' \subseteq X$.
- If Y is a finite configuration contained in X then there exists a finite configuration Y' such that $Y \subset Y' \subseteq X$.
- Suppose Y is a finite configuration contained in X, and $Y \overset{e}{\Rightarrow} Y'$. Suppose that for every p in $loc(a)$, where $a = \lambda(e)$, there exists $Y_p \subseteq X$ such that the p-view of Y_p is identical to the p-view of Y and $Y_p \overset{e1}{\Rightarrow} Y'_p$ with $\lambda(e1) = a$ and $Y'_p \subseteq X$. Then $Y' \subseteq X$.

Now for a distributed strategy *Str*, it is easy to check that the formula *Good*(X) is satisfied under the interpretation that maps X to the set of *Str*-events. Conversely, *Good*(X) is satisfied under the interpretation that maps X to G (where $G \subseteq E$). We can construct a distributed strategy $Str = \{Str_p\}$ from G, where for each p and $\sigma \in L(\mathcal{A})$, $Str_p(\sigma)$ is defined as follows: Let c be the configuration of *ES* that corresponds to $\downarrow^p(\sigma)$, that is, the p-view of σ.

- If $c \subseteq G$, then $Str_p(\sigma)$ is the set given by: $a \in \Sigma_p$ is in $Str_p(\sigma)$ iff there exists $e \in G$ such that $p\text{-}view(\downarrow e - \{e\}) = c$ and $\lambda(e) = a$.
- If c is not a subset of G, then $Str_p(\sigma) = \emptyset$.

Again, it is routine to verify that *Str* as such is a well-defined distributed strategy and the set of *Str*-events is precisely G.

All we need now is to argue that we can assert that every infinite play belonging to a good set meets the specification. But this is easy to do since L_{spec} is robust and is in fact an ω-regular trace language. Hence L_{spec} is definable in the monadic second order logic of infinite traces interpreted over the set of infinite traces generated by our trace alphabet (Σ, I) [18]. Denoting this logical language by $\mathit{MSO}(\Sigma, \mathit{loc})$, we can assume, without loss of generality, that its syntax is *exactly* that of $\mathit{MSO}(\mathit{ES})$ *but* interpreted over infinite traces represented as Σ-labelled partial orders. In particular, the $\lessdot$ refers to the partial order of the trace rather than the positional order of a linearization of the trace.

Now let Φ_{spec} be a sentence in $\mathit{MSO}(\Sigma, \mathit{loc})$ such that the ω-regular trace language defined by it is precisely L_{spec}. For an infinite play σ in $L^{\omega}(\mathcal{A})$, it will be the case that σ in is in L_{spec} iff the Σ-labelled poset $(c, \leq_c, \lambda_c)$ satisfies Φ_{spec} in $\mathit{MSO}(\Sigma, \mathit{loc})$, where c is the unique infinite configuration of $\mathit{ES}_{\mathcal{A}}$ that represents the infinite trace $\{\sigma' \in \Sigma^{\omega} \mid \sigma' \sim \sigma\}$.

We can now construct in $\mathit{MSO}(\mathit{ES})$ a sentence $\exists X.\ \Phi_{\mathit{ctrl}}(X)$, where the formula $\Phi_{\mathit{ctrl}}(X)$ with one free set variable X asserts that X is a good set and moreover, for every infinite configuration Y contained in X, the infinite trace represented by Y, viewed as the Σ-labelled poset $(Y, \leq_Y, \lambda_Y)$, satisfies Φ_{spec}. It is routine to show that the sentence $\exists X.\ \Phi_{\mathit{ctrl}}(X)$ is true in $\mathit{MSO}(\mathit{ES})$ iff there exists a distributed controller for $(\mathcal{A}, L_{\mathit{spec}})$. It now follows from theorem 3.4 that, given the CCP distributed plant $\mathcal{A}$ and the robust specification L_{spec}, one can effectively determine whether there exists a distributed controller for the pair $(\mathcal{A}, L_{\mathit{spec}})$. This establishes theorem 3.9(i).

It follows from theorem 3.8 that the complexity of the decision procedure in theorem 3.9(i) is $\mathit{tower}(O(|\mathit{RS}_{\mathcal{A}}|), O(|\Phi_{\mathit{spec}}|)$, where $|\Phi_{\mathit{spec}}|$ is the size of Φ_{spec} [18].

3.6.2. Branching Time Specifications.

Next we study distributed controller synthesis with branching time specifications. As usual, we fix a CCP plant $\mathcal{A}$. We define a *branching time specification* for $\mathcal{A}$ to be a sentence in $\mathit{MSO}(\mathcal{A})$. In what follows, we fix a branching time specification Ψ_{spec}. One could of course consider branching time specifications given in CTL or CTL* ([21]). We however believe that, in the distributed setting, where executions are partially ordered, it is natural to reason about *non-interleaved* branching time behaviours. In this aspect, the MSO logic over event structures is more suitable than CTL or CTL*, since CTL and CTL* concern interleaved branching time behaviours.

Let *Str* be a distributed strategy for $\mathcal{A}$. Intuitively, we shall say *Str* is winning for Ψ_{spec} iff the "sub-event structure" induced by *Str* is satisfies Ψ_{spec}. We now make this precise. As in the proof of theorem 3.9(i) in section 3.6.1, E_{Str}, the set of *Str*-events, is a downward-closed set of events. Let $ES_{Str} = (E_{Str}, \leq_{Str}, \#_{Str}, \lambda_{Str})$, where $\leq_{Str}$, $\#_{Str}$, λ_{Str} are, respectively, the restrictions of $\leq$, $\#$, λ to E_{Str}. It follows that ES_{Str} is a Σ-labelled event structure. Moreover, the set of formulas in $MSO(ES_{Str})$ is identical with $MSO(ES_{\mathcal{A}})$. The notion of $ES_{Str} \models \Psi_{spec}$ is also clear. Now we say the distributed strategy *Str* is winning for Ψ_{spec} iff $ES_{Str} \models \Psi_{spec}$.

By a distributed controller for the pair $(\mathcal{A},\ \Psi_{spec})$, we shall mean a distributed strategy of $\mathcal{A}$ that is winning for Ψ_{spec}.

Theorem 3.11.

(i) *Given a CCP distributed plant $\mathcal{A}$ and a sentence Ψ_{spec} in $MSO(ES_{\mathcal{A}})$, we can effectively determine whether there exists a distributed controller for $(\mathcal{A},\ \Psi_{spec})$.*

(ii) *Further, if such a distributed controller exists, then we can effectively synthesize a* finite state *one presented in the form of a CCP.*

In what follows, we prove theorem 3.11(i) by an easy modification of the proof of theorem 3.9(i). As is the case with theorem 3.9, the proof of theorem 3.11(ii) requires very different tools from that of theorem 3.11. Hence, we defer the proof of theorem 3.11(ii) to section 3.7.

Proof of theorem 3.11(i). For a set variable Y and a formula ψ in $MSO(ES_{\mathcal{A}})$, we define the *Y-relativized version* of ψ, denoted $Rel_X(\psi)$, inductively as follows:

- Firstly, $Rel_Y(R_a(x)) = R_a(x)$ and $Rel_Y(x \lessdot y) = x \lessdot y$.
- Secondly, we define $Rel_Y(x \in X) = x \in X$. Further, $Rel_Y(\exists x.\,(\Psi)) = \exists x.\, x \in Y \wedge Rel_Y(\Psi)$ and $Rel_Y(\exists X\,(\Psi)) = \exists X.\,(X \subseteq Y) \wedge Rel_Y(\Psi))$.
- Finally, $Rel_Y(\sim \Psi) = \sim Rel_Y(\Psi)$ and $Rel_Y(\Psi 1 \vee \Psi 2) = Rel_Y(\Psi 1) \vee Rel_Y(\Psi 2)$.

Let $Good(X)$ be the predicate as in the proof of theorem 3.9(i). It follows that there exists a distributed controller for $(\mathcal{A}, \Psi_{spec}(X))$ iff the sentence $\exists X.\ Good(X) \wedge Rel_X(\Psi_{spec})$ is true in $MSO(ES_{\mathcal{A}})$. Hence, this establishes theorem 3.11(i), owing to theorem 3.4. □

It follows from theorem 3.8 that the complexity of the decision procedure in theorem 3.11(i) is $tower(O(|RS_{\mathcal{A}}|), O(|\Psi_{spec}|)$, where $|\Psi_{spec}|$ is the size of Ψ_{spec}.

3.7. Synthesis of Finite State Distributed Controllers

The goal of this section is show the effective synthesis of finite state distributed controller. More precisely, we shall prove theorem 3.9(ii) and theorem 3.11(ii).

In what follows, we show the proof of theorem 3.9(ii). It will be clear that theorem 3.11(ii) can be proved in exactly the same way as theorem 3.9(ii).

We fix $\mathcal{A}$ and L_{spec} as in theorem 3.9, and assume a distributed controller for the pair $(\mathcal{A}, L_{spec})$ has been known to exist. Recall from section 3.4 the full Σ-tree TR and also the map $\|\cdot\|$ from $MSO(ES_{\mathcal{A}})$ into $MSO(TR)$.

To establish theorem 3.9(ii), we need two ideas. Firstly, for a sentence ψ in $MSO(TR)$, the decision algorithm for testing the truth of ψ yields a "witness" for ψ in the form of a regular labelled tree, in case ψ is true. Secondly, Zielonka's famous theorem [81] states every regular trace-closed language can be accepted by a deterministic asynchronous automaton [18]. A deterministic asynchronous automaton is basically a deterministic asynchronous transition system with a designated subset of global states as accepting states. A string σ is accepted by a deterministic asynchronous automaton $\mathcal{B}$ iff running $\mathcal{B}$ on σ leads to an accepting global state. However, we are seeking finite state distributed controllers in the form of asynchronous transition system. A deterministic asynchronous transition system may be viewed as an deterministic asynchronous automaton of which every global state is accepting. This complication requires use to appeal to the following variation of Zielonka's theorem:

Proposition 3.12 ([71]). *Let L is a regular trace language over the distributed alphabet (Σ, loc). If L is prefix-closed and for every σ in $\Sigma^\star$, $a, b \in \Sigma$, it is the case that $\sigma ab \in L$ whenever $\sigma a \in L$ and $\sigma b \in L$ and $a\ I\ b$, then one can effectively construct a deterministic asynchronous automaton $\mathcal{C}$ over (Σ, loc) such that the language accepted by $\mathcal{C}$ is L and moreover, every global state of $\mathcal{C}$ is an accepting state.*

Now we are ready to prove theorem 3.9(ii).

Proof of theorem 3.9(ii). Let $\|\exists X.\ \Phi_{ctrl}(X)\| = \exists X.\ \Psi_{ctrl}(X)$, where $\Psi_{ctrl}(X)$ is a formula in $MSO(TR)$ with one free set variable X. By Rabin's result [63] that $MSO(TR)$ is decidable, one can effectively determine whether the sentence $\exists X.\ \Psi_{ctrl}(X)$ holds in TR. This test is in fact performed by first constructing a non-deterministic Rabin tree automaton $\mathcal{R}$ that runs over $\{\top, \bot\}$-labelled Σ-tree whose underlying tree is TR, where $\top$,$\bot$ are special symbols, that has the following property: A $\{\top, \bot\}$-labelled Σ-tree (TR, η) is accepted by $\mathcal{R}$ iff $TR \models_{\mathcal{I}} \Psi_{ctrl}(X)$ where $\mathcal{I}$ maps X to the set of $\top$-labelled nodes, namely, $\{w \in TR \mid \eta(w) = \top\}$. It follows that $\exists X.\ \Psi_{ctrl}(X)$ is true in TR iff the language of $\mathcal{R}$ is nonempty. By Rabin's tree theorem [63], we can effectively test whether the language of $\mathcal{R}$ is nonempty and thus determine the truth of $\exists X.\ \Psi_{ctrl}(X)$. Further, if the language of $\mathcal{R}$ is nonempty, then the nonemptiness testing algorithm also produces a regular $\{\top, \bot\}$-labelled Σ-tree (TR, η). This implies that the set $W_{ctrl} = \{w \in TR \mid \eta(w) = \top\}$ is a regular subset of $\Sigma^\star$ and can be effectively extracted from the finitary presentation of (TR, η).

Clearly W_{ctrl} must be a subset of $LEX_{\mathcal{A}}$. We note that W_{ctrl} can be presented as a finite state automaton and moreover, every state of this finite state automaton is an accepting state.

Let $W'_{ctrl} \subseteq \Sigma^\star$ be given by: $\sigma \in \Sigma^\star$ is in W'_{ctrl} iff there exists σ' in W_{ctrl} such that $\sigma \sim \sigma'$. It follows ([18]) that W'_{ctrl} is a regular subset of $\Sigma^\star$ and is thus a regular trace language over (Σ, I). From W'_{ctrl}, we define the set $\widehat{W}_{ctrl} \subseteq \Sigma^\star$ as follows: $\sigma \in \Sigma^\star$ is in $\widehat{W}_{ctrl}$ iff for every $\sigma' \in \Sigma^\star$ with $[\sigma']$ being a prime trace and $[\sigma'] \sqsubseteq [\sigma]$, it is the case that $\sigma' \in W'_{ctrl}$. Since W'_{ctrl} is a regular trace language over (Σ, I), it is easy to show ([18]) that $\widehat{W}_{ctrl}$ is also a regular trace language over (Σ, I).

We note that W_{ctrl} represents the distributed strategy $Str = \{Str_p\}$ defined as follows:

- For every p, for every $\sigma \in L(\mathcal{A})$, $Str_p(\sigma)$ is given by: $a \in \Sigma_p$ is in $Str_p(\sigma)$ iff there exists $\sigma' a$ in $\Sigma^\star$ such that $\sigma' a \in W_{ctrl}$ and $\downarrow^p(\sigma') \sim \downarrow^p(\sigma)$.

It is routine to verify that Str is well-defined. Further, the set of events $\{e = [\sigma] \in ES \mid lin([\sigma]) \in W_{ctrl}\}$ is the set of Str-events; and $L(Str)$ is precisely $\widehat{W}_{ctrl}$.

It is straightforward to verify that $\widehat{W}_{ctrl}$ satisfies the conditions set out in proposition 3.12, since Str is *distributed*. It follows that we can effectively construct a deterministic ATS $\mathcal{C}$ such that $L(\mathcal{C})$ is $\widehat{W}_{ctrl}$.

Note that $\mathcal{A}$ is a CCP and $\widehat{W}_{ctrl}$ is a subset of $L(\mathcal{A})$. Hence, following the definition of connectedly communicating, it is routine to check that $\mathcal{C}$ must be a CCP. This completes the proof of theorem 3.9. $\square$

3.8. Undecidability Results

In this section, we present some undecidability results concerning distributed controller synthesis for CCPs.

3.8.1. Non-Robust Linear Time Specifications.

We begin by considering non-robust linear time specifications.

Theorem 3.13. *There is no effective procedure which can, given a CCP distributed plant $\mathcal{A}$ and a specification L_{spec} (that is not necessarily robust), determine whether there exists a distributed controller for $(\mathcal{A}, L_{spec})$.*

As with many undecidability results in distributed controller synthesis, the proof of theorem 3.13 relies on the undecidability proof of the multi-player game with partial information developed in [60]. In what follows, we prove theorem 3.13 by a reduction from the *two-player-one-adversary (2-P-1-A) game problem* which is known to be undecidable following the results in [60].

A *2-P-1-A alphabet* is a structure $\widehat{\Gamma} = (\Gamma_1^{env}, \Gamma_1^{sys}, \Gamma_2^{env}, \Gamma_2^{sys})$ where Γ_1^{env}, Γ_1^{sys}, Γ_2^{env}, Γ_2^{sys} are disjoint finite alphabets. An instance of the 2-P-1-A game problem over $\widehat{\Gamma}$ is a subset G_{spec} of $(\Gamma_1^{env}.\Gamma_1^{sys}.\Gamma_2^{env}.\Gamma_2^{sys})^\omega$ such that G_{spec} is ω-regular (over $\Gamma_1^{env} \bigcup \Gamma_1^{sys} \bigcup \Gamma_2^{env} \bigcup \Gamma_2^{sys}$). The 2-P-1-A game is played in infinitely many rounds. Each round consists of four moves: firstly, the adversary picks a letter in Γ_1^{env}; secondly, player 1 picks a letter in Γ_1^{sys}; thirdly the adversary picks a letter in Γ_2^{env}; finally, player 2 picks a letter in Γ_2^{sys}. A 2-P-1-A strategy is a pair (f_1, f_2) where f_i is the function from $(\Gamma_i^{env}.\Gamma_i^{sys})^\star.\Gamma^{env}$ to Γ_i^{sys}. Let σ be in $(\Gamma_1^{env}.\Gamma_1^{sys}.\Gamma_2^{env}.\Gamma_2^{sys})^\omega$. We say that σ is according to the 2-P-1-A strategy (f_1, f_2) iff for every prefix $\tau a_1 b_1 a_2 b_2$ of σ, where $\tau \in (\Gamma_1^{env}.\Gamma_1^{sys}.\Gamma_2^{env}.\Gamma_2^{sys})^\star$, $a_1 \in \Gamma_1^{env}$, $b_1 \in \Gamma_1^{sys}$, $a_2 \in \Gamma_2^{env}$, $b_2 \in \Gamma_2^{sys}$, it is the case that $b_i = f_i(\tau_i a_i)$ with $\tau_i = \tau \upharpoonright (\Gamma_i^{env} \bigcup \Gamma_i^{sys})$ for $i = 1, 2$. We note that player i only knows the history of actions in the alphabets Γ_i^{env}, Γ_i^{sys}. The 2-P-1-A strategy (f_1, f_2) is winning for G_{spec} iff the following holds: for every σ in $(\Gamma_1^{env}.\Gamma_1^{sys}.\Gamma_2^{env}.\Gamma_2^{sys})^\omega$, if σ is according to (f_1, f_2), then σ is in G_{spec}.

The proposition below follows from results in [60].

Proposition 3.14 ([60]). *One can construct a 2-P-1-A alphabet $\widehat{\Gamma} = (\Gamma_1^{env}, \Gamma_1^{sys}, \Gamma_2^{env}, \Gamma_2^{sys})$ such that there is no effective procedure which can, given an 2-P-1-A instance G_{spec} over $\widehat{\Gamma}$, determine whether there exists a winning 2-P-1-A strategy for G_{spec}.*

Proof. The proof is via a reduction from the halting problem for Turing machines. Intuitively, the alphabet Γ_1^{sys} contains letters suitable for describing configurations of a Turing machine and a marker symbol \$. The alphabet Γ_1^{env} contains letters to tell player 1 whether to generate the marker symbol \$ or to generate letters representing Turing machine configurations. The alphabets Γ_2^{env}, Γ_2^{sys} have similar meanings.

The intuition of the reduction is as follows. We shall use environment moves to force each player to output configurations of a Turing machine. The team of two players wins iff each player outputs a *successive* sequence of configurations ending at a halting configuration. This condition can be checked by interleaving the sequence of configurations produced by player 1 with that produced by player 2. The crucial point to note is that successive configurations of a Turing machine differ only by boundedly many letters.

Given a Turing machine $\mathcal{M}$, we construct a 2-P-1-A game instance G_{spec} such that for every σ in $(\Gamma_1^{env}.\Gamma_1^{sys}.\Gamma_2^{env}.\Gamma_2^{sys})^\omega$, σ is in G_{spec} iff $\sigma \upharpoonright (\Gamma_1^{env} \bigcup \Gamma_1^{sys})$ and $\sigma \upharpoonright (\Gamma_2^{env} \bigcup \Gamma_2^{sys})$ both "represent" the same infinite sequence of configurations of the form $\xi = C_1 C_2 \ldots C_m C_m C_m \ldots$, where C_1 is an initial configuration; and for $i = 1, 2, \ldots, m-1$, $\mathcal{M}$ can go from C_i to C_{i+1} in one move; and C_m is a halting configuration. The required condition on σ can be checked by a non-deterministic Büchi automaton which reads configurations of $\sigma \upharpoonright (\Gamma_1^{env} \bigcup \Gamma_1^{sys})$ and $\sigma \upharpoonright (\Gamma_2^{env} \bigcup \Gamma_2^{sys})$ in an interleaving fashion. Hence

G_{spec} is a well-defined 2-P-1-A problem instance. We refer to [60, 62] for detailed arguments. $\square$

We now prove theorem 3.13.

Proof of theorem 3.13. Let $\widehat{\Gamma} = (\Gamma_1^{env}, \Gamma_1^{sys}, \Gamma_2^{env}, \Gamma_2^{sys})$ be the 2-P-1-A alphabet constructed in Proposition 3.14. Let G_{spec} be an instance of the 2-P-1-A game problem over $\widehat{\Gamma}$. We construct a CCP distributed plant $\mathcal{A}$ and a specification L_{spec} such that there exists a winning 2-P-1-A strategy for G_{spec} iff there exists a distributed controller for $(\mathcal{A}, L_{spec})$. This will then establish theorem 3.13.

Let $\mathcal{P} = \{p_1, p_2\}$. Set (Σ, loc) to be the distributed alphabet where $\Sigma = \Gamma_1^{env} \bigcup \Gamma_1^{sys} \bigcup \Gamma_2^{env} \bigcup \Gamma_2^{sys}$ and $loc(a) = \{p_1\}$ for every $a \in \Gamma_1^{env} \bigcup \Gamma_1^{sys}$ and $loc(a) = \{p_2\}$ for every $a \in \Gamma_2^{env} \bigcup \Gamma_2^{sys}$. The distributed plant $\mathcal{A}$ will be over (Σ, loc). Figure 3.5 illustrates intuitive idea of $\mathcal{A}$. More precisely, we have $\mathcal{A} = (\{S_{p_1}^{env}, S_{p_2}^{env}\}, \{S_{p_1}^{sys}, S_{p_2}^{sys}\}, s_{in}, \Sigma^{env}, \Sigma^{sys}, \{\delta_a\}_{a\in\Sigma})$ where $S_{p_i}^{env} = \{s_{p_i}^{env}\}$ for $i = 1, 2$; $S_{p_i}^{sys} = \{s_{p_i}^{sys}\}$ for $i = 1, 2$; $s_{in} = (s_{p_1}^{env}, s_{p_2}^{env})$; $\Sigma^{env} = \Gamma_1^{env} \bigcup \Gamma_2^{env}$; $\Sigma^{sys} = \Gamma_1^{sys} \bigcup \Gamma_2^{sys}$. And for $i = 1, 2$, we have $\delta_a = \{(s_{p_i}^{env}, s_{p_i}^{sys})\}$ if $a \in \Gamma_i^{env}$; $\delta_a = \{(s_{p_i}^{sys}, s_{p_i}^{env})\}$ if $a \in \Gamma_i^{sys}$.

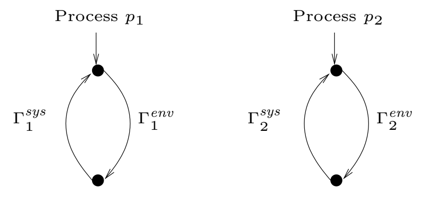


FIGURE 3.5

We define L_{spec} as follows: $\sigma \in \Sigma^\omega$ is in L_{spec} iff either σ is not in $(\Gamma_1^{env}.\Gamma_1^{sys}.\Gamma_2^{env}.\Gamma_2^{sys})^\omega$, or σ is in G_{spec}. It is now routine to verify that there exists a winning 2-P-1-A strategy for G_{spec} iff there exists a distributed controller for $(\mathcal{A}, L_{spec})$. $\square$

3.8.2. Strictly Local Strategies.

One can also study distributed controller synthesis for distributed plants with respect to *strictly local* strategies. Let $\mathcal{A}$ be a distributed plant. We say that a p-strategy f for $\mathcal{A}$ is *strictly local* if it satisfies: for every $\sigma, \sigma' \in L(\mathcal{A})$, $\sigma \upharpoonright \Sigma_p = \sigma' \upharpoonright \Sigma_p$ implies $f(\sigma) = f(\sigma')$. Hence a strictly local p-strategy depends only on the projection of the play onto Σ_p.

Let L_{spec} be a specification. By a *strict distributed controller* for the pair $(\mathcal{A}, L_{spec})$, we mean a family $\{Str_p\}$ of strictly local p-strategies for $\mathcal{A}$, one for each p; and $\{Str_p\}$ is winning for L_{spec}. We have:

Theorem 3.15. *There is no effective procedure which can, given a CCP distributed plant $\mathcal{A}$ and a robust specification L_{spec}, determine whether there exists a strict distributed controller for $(\mathcal{A}, L_{spec})$.*

Proof. Let $\widehat{\Gamma} = (\Gamma_1^{env}, \Gamma_1^{sys}, \Gamma_2^{env}, \Gamma_2^{sys})$ be the 2-P-1-A alphabet constructed in Proposition 3.14. Let G_{spec} be an instance of the 2-P-1-A game problem over $\widehat{\Gamma}$. We construct a CCP distributed plant $\mathcal{A}$ and a *robust* specification L_{spec} such that there exists a winning 2-P-1-A strategy for G_{spec} iff there exists a strict distributed controller for $(\mathcal{A}, L_{spec})$. This will then establish theorem 3.15.

Let $\mathcal{P} = \{p_1, p_2\}$. Set (Σ, loc) to be the distributed alphabet where $\Sigma = \Gamma_1^{env} \bigcup \Gamma_1^{sys} \bigcup \Gamma_2^{env} \bigcup \Gamma_2^{sys} \bigcup \{\$, \$\$\}$; and $loc(a) = \{p_1\}$ for every $a \in \Gamma_1^{env} \bigcup \Gamma_1^{sys}$, $loc(a) = \{p_2\}$ for every $a \in \Gamma_2^{env} \bigcup \Gamma_2^{sys}$, $loc(\$) = \{p_1, p_2\} = loc(\$\$)$. The distributed plant $\mathcal{A}$ will be over (Σ, loc). Figure 3.6 illustrates intuitive idea of $\mathcal{A}$. More precisely, we have $\mathcal{A} = (\{S_{p_1}^{env}, S_{p_2}^{env}\}, \{S_{p_1}^{sys}, S_{p_2}^{sys}\}, s_{in}, \Sigma^{env}, \Sigma^{sys}, \{\delta_a\}_{a\in\Sigma})$ where $S_{p_i}^{env} = \{s_{p_i}^{env}\}$ for $i = 1, 2$; $S_{p_i}^{sys} = \{s_{p_i}^{sys}, s_{p_i}^{\$}, s_{p_i}^{\$\$}\}$ for $i = 1, 2$; $s_{in} = (s_{p_1}^{env}, s_{p_2}^{\$})$; $\Sigma^{env} = \Gamma_1^{env} \bigcup \Gamma_2^{env}$; $\Sigma^{sys} = \Gamma_1^{sys} \bigcup \Gamma_2^{sys} \bigcup \{\$, \$\$\}$, and $\{\delta_a\}_{a\in\Sigma}$ is given as follows. For $i = 1, 2$, if $a \in \Gamma_i^{env}$, then $\delta_a = \{(s_{p_i}^{env}, s_{p_i}^{sys})\}$. For $a \in \Gamma_1^{sys}$, we have $\delta_a = \{(s_{p_1}^{sys}, s_{p_1}^{\$}\}$. For $a \in \Gamma_2^{sys}$, we have $\delta_a = \{(s^{sys}_{p_2}, s^{\$\$}_{p_2})\}$. Finally, $\delta_{\$} = \{((s^{sys}_{p_1}, s^{\$}_{p_2}), (s^{\$\$}_{p_1}, s^{env}_{p_2}))\}$, and $\delta_{\$\$} = \{((s^{\$\$}_{p_1}, s^{\$\$}_{p_2}), (s^{env}_{p_1}, s^{\$}_{p_2}))\}$. Note that $L^\omega(\mathcal{A})$ is precisely the set

$$(\Gamma^{env}_1.\Gamma^{sys}_1.\{\$\}.\Gamma^{env}_2.\Gamma^{sys}_2.\{\$\$\})^\omega \ .$$

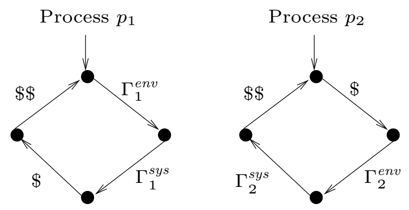


FIGURE 3.6

We define L_{spec} as follows: $\sigma \in \Sigma^\omega$ is in L_{spec} iff σ is in $L^\omega(\mathcal{A})$ and the projection of σ onto $\Gamma^{env}_1 \bigcup \Gamma^{sys}_1 \bigcup \Gamma^{env}_2 \bigcup \Gamma^{sys}_2$ is in G_{spec}. It is easy to see that L_{spec} is robust. It is also routine to verify that there exists a winning 2-P-1-A strategy for G_{spec} iff there exists a strict distributed controller for $(\mathcal{A}, L_{spec})$. $\square$

3.9. Discussion

One can also assume that the distributed plant itself is not a CCP but require, for robust specifications, the distributed controller be a CCP. More precisely, we say that the distributed strategy *Str* is *k-communicating* iff for every $\sigma \in L(Str)$, if $\sigma\sigma' \in L(Str)$ and $|\sigma'|_p \geq k$ and $|\sigma'|_q = 0$, then for every $\sigma\sigma'\sigma'' \in L(Str)$, p, q are separated in σ''. We say *Str* is *connectedly communicating* iff *Str* is k-communicating for some integer k. The *k-communicating distributed controller synthesis* is: Given a distributed plant $\mathcal{A}$, that is not necessarily a CCP, and a robust linear time specification L_{spec} and an integer k, does there exists a k-communicating distributed controller for $(\mathcal{A}, L_{spec})$? We conjecture that the k-communicating controller synthesis problem is decidable; and in case such a distributed controller exists, a finite state one exists as well and it can be effectively synthesized. It is also interesting to study *the connectedly communicating controller synthesis problem*: Given a distributed plant, that is not necessarily connectedly communicating, and a robust specification, does there exist a connectedly communicating distributed controller? We conjecture that the connectedly communicating controller synthesis problem is undecidable.

CHAPTER 4

Controller Synthesis for Real-Time Systems with Tasks

In this chapter, we investigate controller synthesis in the real-time setting. Our aim is to study the problem of synthesizing admission controllers for real-time systems with tasks. We begin with an overview of this problem in section 4.1. Subsequently we discuss related work in section 4.2. In section 4.3, we formulate the plant model that represents the arrival pattern of tasks in a real-time system. And we define the admission controller synthesis problem in section 4.4. Section 4.5 presents our results on the admission controller synthesis problem. We prove that the admission controller synthesis is decidable for quality-of-service specifications given in LTL (linear time temporal logic), and more generally in QPLTL (quantified propositional LTL). We show further that if there exists an admission controller, then we can synthesize one in the form of a finite timed automaton. In section 4.6 we discuss the prospects for extending the current work.

4.1. Overview

In many real-time systems, there are *hard* tasks that are time-critical and *soft* tasks that are not. A soft task might be discarded causing only performance penalties, while a hard task must be always be served, that is, put into the ready queue, and must be completed before its deadline.

With fixed computing resources and a fixed scheduling policy, the task arrival pattern of a real-time system may be such that some tasks will miss their deadlines, if every released task instance is put into the ready queue. In other words, the task arrival pattern may *not* be schedulable. An approach to deal with this is proposed in [45]. For each new task instance, we check if the ready queue would still be schedulable if the new task instance were to be served, that is, put into the ready queue according to the scheduling policy. This checking is the called the *acceptance test*. Each new task instance will be added to the ready queue only if it passes an acceptance test. In particular, [45] discusses ways of designing the acceptance test for various restricted kind of task arrival patterns. However, this approach is not satisfactory for the following reason. It may happen that a time-critical task instance fails the acceptance test, simply because we have admitted earlier too many soft task instances.

We address this problem in a systematic manner and term it the *admission controller synthesis* problem. Informally, given the task arrival pattern of a real-time system and a quality-of-service (QoS) requirement, we would like to determine whether there exists an admission controller. The admission controller will, upon each newly arrived task instance, either accept it and put it into the ready queue through the scheduling policy, or reject (discard) it. We require that every accepted task instance must be completed before its deadline, and moreover, the actions of the admission controller must meet the QoS requirement. In particular, with suitable QoS requirements, we will prevent the admission controller from simply rejecting all soft task instances.

We assume a uniprocessor setting with the preemptive EDF (earliest-deadline-first) scheduling policy. The preemptive EDF policy is known to be optimal in the uniprocessor setting [15] in the following sense: If a task set is schedulable at all, then it is schedulable under the preemptive EDF policy. In the sequel, we discuss the choice of formalisms for modelling task arrival patterns and QoS requirements.

Classical schedulability analysis techniques [15] for real-time systems make strong assumptions about the temporal arrival patterns of the tasks.

To overcome this limitation, a model was suggested in [24], where timed automata [7] are used to describe task arrival patterns. In this way, many task arrival patterns can be captured in a uniform way. It was shown in [24] that, with a uniprocessor setting and the preemptive EDF scheduling policy, one can effectively decide whether all tasks can be scheduled to meet their deadlines when the task arrivals are described by a timed automaton.

We shall adopt the approach in [24] of modelling task arrival patterns using timed automata. Further, we extend such timed automata slightly to form what we called *task plants*. A task plant will have environment states from which the environment can make uncontrollable timed moves to release tasks. Each such uncontrollable action, leading to a system state, will then be immediately followed by an urgent pair of controllable actions; one of them, *accepting* the just released task instance and putting it into the ready queue and the other one *rejecting* it. The admission controller synthesis problem is then to determine whether there exists an admission strategy for choosing the controllable actions so that the admitted task instances, no matter what the environment does, can all be scheduled without missing their deadlines. And moreover the task acceptance pattern by the admission strategy satisfies a given QoS requirement.

We shall label each transition of the task plant with a set of atomic propositions and consider a QoS requirement to be given as a formula in LTL, or more generally in QPLTL (cf. section 2.2). For such a formula ψ, we say an admission controller satisfies ψ iff along every infinite run σ of the task plant that can be generated by the admission controller, the sequence of sets of atomic positions induced by σ is a model of ψ.

The admission controller synthesis problem can be more precisely stated: Given a task plant and a QoS specification in LTL or in QPLTL, does there exist an admission controller? We prove that this problem is decidable, for both QoS specifications in LTL and in QPLTL. Further, in both cases, we show that if an admission controller exists, then we can effectively synthesize one in the form of a (finite) timed automaton.

In the sequel, we elaborate on the kind of QoS requirements that can be expressed with this framework.

Obviously, in LTL, we can easily indicate that a task τ is hard by asserting that every instance of τ must be accepted. Hence in the task plant model, we need not impose a syntactic distinction between hard and soft tasks. We can also express in LTL liveness and fairness properties, which may be seen as QoS requirements of *qualitative* nature. For instance, one can specify the liveness property that along every infinite run, if instances of task τ is released infinitely often, then infinitely often some instance of τ must be accepted. One can also demand the fairness property that assuming instances of task τ and τ' are released infinitely often, if infinitely often some instance of τ is accepted, then infinitely often some instance of τ' is accepted.

More interestingly, we can express in LTL *quantitative* QoS requirements that have the "boundedness" flavour. For instance, for a *fixed* integer n and a task τ, we can construct an LTL formula φ to assert that between every consecutive n instances of τ, at least $0.7n$ of them must be accepted. Specifically, we will enumerate all possible acceptance patterns of n consecutive arrivals of instances of τ in which more than $0.7n$ instances are accepted, and define φ will be a disjunction of all these acceptance patterns.

For a fixed integer n, we can specify in QPLTL the QoS requirement Ψ_n that: every n-th instance of task τ must be accepted, while other instances of τ may or may not be accepted. Such a property is not expressible in LTL [80]. We refer to section 2.2 for the construction of Ψ_n.

On the other hand, it seems we can not express in LTL or QPLTL quantitative QoS requirements that concern the limit average behaviour of task acceptance patterns. Such a property may demand that the *limit* of the average acceptance ratio of instances of task type τ is at least 0.7. It would be worthwhile to extend our work to handle such quantitative QoS requirements. For doing so, we believe that techniques from quantitative games (see for instance [20, 82]) will turn out to be useful.

Our work in admission controller synthesis may be viewed as an extension to an open system framework of the results reported in [24]. As discussed already, this extension has a natural motivation and it is a pleasing fact that techniques from the controller synthesis domain and timed-automata-based schedulability analysis techniques can be combined in a natural manner to solve the synthesis problem at hand.

4.2. Related Work

In the literature, a number of studies are available regarding controller synthesis in a timed setting; a representative sample being [9, 13, 19, 53]. The key motivation of these works is to extend classical controller synthesis results for discrete event systems [14, 61, 66] to a timed setting. In comparison, though we use the language and techniques of (timed) controller synthesis, our motivation is very different. Our goal is to derive admission controller for real-time systems with tasks so as to obtain schedulability and to satisfy QoS requirements. We chose to study real-time systems with tasks, as we believe that the correctness of many real-time systems depends not only on timely occurrence of events, but also the proper handling of tasks triggered by these events.

In particular, the work [9] studied timed games played on timed automata with the safety, denoted $\Box$, winning condition which asserts that the controlled timed plant only visits the good states. It also considers dually, the eventuality, denoted $\Diamond$, winning condition which dictates that the controlled timed plant will eventually reach one of the good states. The emphasis of [9] is to extend efficient symbolic methods for analysis of timed systems to derive *practically efficient* synthesis procedures for solving timed games.

The work [53] extends the results of [9] to include also $\Diamond\Box$ and $\Box\Diamond$ and Rabin winning conditions. The $\Diamond\Box$ condition asserts that the set of states that the controlled timed plant will visit infinitely often are contained in a prescribed subset of good states, while the $\Box\Diamond$ condition demands that some state from a prescribed subset of good states is visited infinitely often by the controlled timed plant. The Rabin condition is a sequence of accepting pairs $\{(F_i, G_i) \mid i = 1, 2, \ldots, k\}$ where F_i,G_i are prescribed subsets of good states. This requires that, for some accepting pair (F_i, G_i), the controlled timed plant visits some state from F_i infinitely often and moreover the set of states that the controlled timed plant visited infinitely often is contained in G_i.

The work [19] studies timed games where the plant is based on a timed automaton and also the specification is given as another timed automaton. In this sense, the specification is *external* to the plant. In view that classical controller synthesis on discrete event systems from the control theory literature often deals with *internal* specification (in terms of the plant states), the objective of [19] is to study the effect of external specifications on decidability and undecidability of timed games.

The work [13] carries on the framework of [19] and investigates decidability and undecidability results of timed games for both internal and external specifications, but with partial observation. The partial observation complication dictates that the controllers are not able to observe all the actions of the environment.

Both [13, 19] also consider whether the clocks available to the controller and the granularity of these clocks are fixed a prior or can be chosen by the controller. In our setting of admission controller synthesis, the admission controller will not have any clock variables of its own. It will be perhaps interesting to extend our controller synthesis to settings where the admission controller is endowed with its own clocks and granularity.

A second line of work related to admission controller synthesis is to derive a schedule for a real-time application, given the timed model of the application and a set of resource constraints [1, 5, 10, 12, 28, 41, 58] with [1, 5] in fact carrying out the work using the controller synthesis paradigm. The emphasis in this line of work however is to restrict the timed behaviours of the application so as to meet, in a timely fashion, access to shared resources. At present, we have considered a uniprocessor setting. It will be interesting to extend our work along this line, to multi-processor settings accompanied by resource access protocols for shared resources.

In particular, the work [1] investigates the computation of optimal schedules for a given set of tasks that require shared resources. The execution time of a task is not fixed and rather varies in a given interval. It is shown in [1] that the problem can be reduced to synthesizing controllers for timed automata with reachability winning conditions. More generally, the work [5] studies the following problem: given a timed automaton modelling a real-time system with tasks and a constraint dictating the timing properties of the tasks (called processes in [5]) and policy requirements about resource management (dynamic priorities, preemption, etc), one seeks a scheduler such that the tasks are schedulable (deadlines are not violated) while respecting the policy requirements about resource management.

4.3. The Task Plant Model

In this section, we formulate the task plant model and define its operational semantics.

4.3.1. Timed Automata.

Since our task plant model will be based on timed automata. We begin by reviewing the basics of timed automata. For detailed reference, we refer to [7].

A timed automaton is basically a finite transition system augmented with *clocks*. Let X be a finite set of clocks. A clock constraint over X is a finite conjunction of basic constraints (inequalities) of the form $x \prec c$ or $x - y \prec c$ where $x, y \in X$, $c \in \mathbb{N}$, $\prec \in \{<, \leq, >, \geq\}$. We let $Grd(X)$ denote the set of clock constraints over X. As usual $\mathbb{N}$ is the set of natural numbers. The difference constraints $x - y \prec c$ will be needed later to capture the behaviour of the ready queue of tasks. Formally, a timed automaton $\mathcal{A}$ is a structure $(Q, q_{in}, X, \{I_q\}_{q \in Q}, \Sigma, \longrightarrow)$ where Q is a finite set of locations and $q_{in} \in Q$ is the initial location. X is a finite set of *clocks*. For each $q \in Q$, $I_q \in Grd(X)$ is the invariant associated with q. Σ is a finite set of events. And $\longrightarrow \subseteq Q \times Grd(X) \times \Sigma \times 2^X \times Q$ is the transition relation.

Suppose $\mathcal{A}_1 = (Q_1, q^1_{in}, X_1, \{I^1_q\}_{q \in Q_1}, \Sigma, \longrightarrow_1)$ and $\mathcal{A}_2 = (Q_2, q^2_{in}, X_2, \{I^2_q\}_{q \in Q_2}, \Sigma, \longrightarrow_2)$ are timed automata, where X_1 and X_2 are disjoint. Then the product of $\mathcal{A}_1$ and $\mathcal{A}_2$ is the timed automaton $\mathcal{A} = (Q, q_{in}, X, \{I_q\}_{q \in q}, \Sigma, \longrightarrow)$ where $Q = Q_1 \times Q_2$, $q_{in} = (q^1_{in}, q^2_{in})$, $X = X_1 \cup X_2$. And for each $(q1, q2) \in Q_1 \times Q_2$, $I_{(q1,q2)} = I_{q1} \wedge I_{q2}$. Finally, $\longrightarrow$ is the least set such that: if $q1 \overset{\varphi_1, a, Y_1}{\longrightarrow}_1 q1'$ and $q2 \overset{\varphi_2, a, Y_2}{\longrightarrow}_2 q2'$, then $(q1, q2) \overset{\varphi, a, Y}{\longrightarrow} (q1', q2')$, where $\varphi = \varphi_1 \wedge \varphi_2$ and $Y = Y_1 \cup Y_2$.

In what follows, $\mathbb{R}_{\geq 0}$ and $\mathbb{R}_+$ will denote the set of non-negative reals and positive reals, respectively. A clock valuation V over X is a function $X \to \mathbb{R}_{\geq 0}$. For $t \in \mathbb{R}_+$, $V + t$ is the clock valuation $(V + t)(x) = V(x) + t$ for $x \in X$. For $Y \subseteq X$, $V[Y := 0]$ is the clock valuation which maps every clock in Y to zero and agrees with V on other clocks. The notation that the valuation V satisfies the clock constraint φ is defined in the obvious way.

The timed behaviour of $\mathcal{A}$ is given by the transition system $TS_{\mathcal{A}} = (RC_{\mathcal{A}}, (q_{in}, V_{in}), \mathbb{R}_+ \times \Sigma, \Longrightarrow_{\mathcal{A}})$ where $RC_{\mathcal{A}}$ and $\Longrightarrow_{\mathcal{A}}$ are the least sets satisfying the following.

- Firstly, $(q_{in}, V_{in}) \in RC_{\mathcal{A}}$, where $V_{in}(x) = 0$ for every clock variable x.
- Secondly, suppose $(q, V) \in RC_{\mathcal{A}}$. Suppose further, there exists a transition $(q \xrightarrow{\varphi,a,Y} q')$ and $t \in \mathbb{R}_+$ such that $V + t'$ satisfies I_q for all $t' \in \mathbb{R}_+$ with $t' \le t$, and $V + t$ satisfies both φ and $I_{q'}$. Then $(q', V') \in RC_{\mathcal{A}}$, where $V' = (V+t)[Y := 0]$, and $(q, V) \overset{t,a}{\Longrightarrow}_{\mathcal{A}} (q', V')$.

It is well-known [7] that we can quotient $TS_{\mathcal{A}}$ into a finite transition system, called the *region automaton* $RA_{\mathcal{A}}$ of $\mathcal{A}$. For $x \in X$, let c_x be the maximum constant which appears in basic constraints (of transition guards of) $\mathcal{A}$ of the form $x \prec c$, where $c \in \mathbb{N}$, $\prec$ is in $\{<, \le, >, \ge\}$. We say two clock valuations V and V' are *region-equivalent*, denoted $V \sim V'$ iff the following conditions hold:

- For each $x \in X$, either $\lfloor V(x) \rfloor = \lfloor V'(x) \rfloor \le c_x$; or $V(x) > c_x$ and $V'(x) > c_x$. Further, in the former case, $\text{fra}(V(x)) = 0$ iff $\text{fra}(V'(x)) = 0$, where $\text{fra}(v)$ is the fractional part of v.
- For each $x, y \in X$ such that $V(x) \le c_x$, $V'(x) \le c_x$, $V(y) \le c_y$, and $V'(y) \le c_y$, we have $\text{fra}(V(x)) \le \text{fra}(V(y))$ iff $\text{fra}(V'(x)) \le \text{fra}(V'(y))$.
- For every difference constraint $x-y \prec c$ which appears in (transition guards of) $\mathcal{A}$, V satisfies $x - y \prec c$ iff V' satisfies $x - y \prec c$.

A *clock region* is an equivalence class of $\sim$ (over the set of clock valuations over X). It is clear that a clock region R can be effectively represented as θ, a conjunction of clock constraints of the form $x = c$, $c - 1 < x < c$, $x > c_x$, $x - y = c$, $c - 1 < x - y < c$ and $x - y \prec c'$, where $x, y \in X$, $c \in \mathbb{N}$, $c \le c_x$, and $x - y \prec c'$ is a difference constraint which appears in $\mathcal{A}$. More precisely, for every clock valuation V, V is in R iff V satisfies θ.

We extend $\sim$ to $RC_{\mathcal{A}}$ (denoted also as $\sim$) via: $(q, V) \sim (q', V')$ iff $q = q'$ and $V \sim V'$. A *region* is an equivalence class of $\sim$ over $RC_{\mathcal{A}}$. The key property of regions is the following:

Proposition 4.1. *Let $RC_{\mathcal{A}}$ be as given above and assume the associated notations. If $(q, V) \overset{t,a}{\Longrightarrow}_{\mathcal{A}} (q', V')$ and $(q, V) \sim (q1, V1)$, then there exists $t' \in \mathbb{R}_+$ such that $(q1, V1) \overset{t',a}{\Longrightarrow}_{\mathcal{A}} (q1', V1')$ and $(q1', V1') \sim (q', V')$.*

For (q, V) in $RC_{\mathcal{A}}$, we shall denote the region containing (q, V) by $[(q, V)]_{\sim}$ and often just write $[(q, V)]$. We can now define the region automaton $RA_{\mathcal{A}} = (RG_{\mathcal{A}}, [(q_{in}, V_{in})], \Sigma, \rightsquigarrow_{\mathcal{A}})$, where $RG_{\mathcal{A}}$ is the set of regions, and $\rightsquigarrow_{\mathcal{A}} \subseteq RG_{\mathcal{A}} \times \Sigma \times RG_{\mathcal{A}}$ is given by: $[(q, V)] \overset{a}{\rightsquigarrow}_{\mathcal{A}} [(q', V')]$ iff there exists $(q1, V1)$ in $[(q, V)]$ and there exists $(q1', V1')$ in $[(q', V')]$ such that $(q1, V1) \overset{t,a}{\rightsquigarrow}_{\mathcal{A}} (q1', V1')$ for some $t \in \mathbb{R}_+$. It is easy to see that $RA_{\mathcal{A}}$ is well-defined and can be effectively constructed from $\mathcal{A}$.

4.3.2. The Task Plant Model.

Next we recall how task arrival patterns in a real time environment can be modelled using timed automata as proposed in [24]. The basic idea is to associate a task with each location. Whenever a location is entered, an instance of the task associated with the location is supposed to be released. Here, it will be convenient to associate tasks with the transitions rather than with the locations. We also wish to highlight that we are dealing with an *open system* model—called a *task plant* in the present context—of task arrival patterns on which an admission control policy can be imposed.

Formally, we define a *task plant* $\mathcal{A}$ to be a structure $(Q_e, Q_s, q_{in}, X, \{I_q\}_{q \in Q_e}, \Upsilon, C, D, \longrightarrow_e, \longrightarrow_s)$, where

- Q_e and Q_s are disjoint finite nonempty sets of *environment states* and *system states*, respectively.
- $q_{in} \in Q_e$ is the *initial state.*
- X is a finite set of clocks.

- For each $q \in Q_e$, $I_q \in Grd(X)$, is the *invariant* associated with q.
- Υ is a finite set of *task types*. The functions $C, D : \Upsilon \to \mathbb{N}$ associate with each task type a *computation time* and a *relative deadline*, respectively. Further, for each $\tau \in \Upsilon$, $0 < C(\tau) \leq D(\tau)$.
- $\longrightarrow_e \subseteq Q_e \times Grd(X) \times \Upsilon \times 2^X \times Q_s$ is a set of *environment transitions*. For each system state q', there exists a *unique* environment state q and a *unique* environment transition of the form $(q \xrightarrow{\varphi,\tau,Y}_e q')$.
- $\longrightarrow_s \subseteq Q_s \times \{0,1\} \times Q_e$ is a set of *system transitions*. For each system state q', there exists a *unique* environment state q and exactly two system transitions of the form $(q' \xrightarrow{0}_s q)$ and $(q' \xrightarrow{1}_s q)$.

Through the rest of this chapter, we fix a task plant $\mathcal{A}$ defined above and assume the associated notations and terminologies. Informally, the task plant model consists of a timed automaton whose events are interpreted as tasks, in case they are associated with environment transitions. For system transitions we allow only the events $\{0,1\}$ which will be used to capture the decisions made by the controller. The semantics of the task plant will implicitly impose a *zero-delay* on the system states. In other words, as soon as a system state is entered, the admission controller will make the decision to either accept the task that has just been released by the environment; this is captured by the 1-labelled transition going out of the system state. On the other hand, the 0-labelled transition going out of a system state models the decision to reject the just released task. We could have assigned a special clock variable to capture the immediacy of these transitions, but we have not done so for convenience. As mentioned above, the decision as to whether a just released task is to be admitted or not is made as soon as the task is released by the environment. Thus system moves come in pairs and each such pair is uniquely associated with an environment move. Further, the environment is oblivious to the admission policy being followed by the system. This explains the restrictions placed on the structure of the transitions. An example of a task plant is shown in figure 4.1 below. Environment states are indicated by circles and system states by boxes. We have $C(\tau_a) = 1$, $D(\tau_a) = 2$, $C(\tau_b) = 2$, $D(\tau_b) = 3$. All invariants associated with environment states are true.

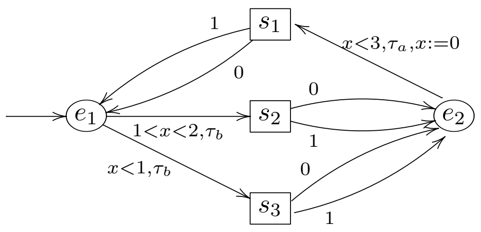


FIGURE 4.1. A task plant

4.3.3. The Ready Queue States.

The semantics of the task plant $\mathcal{A}$ is to be understood with respect to a *scheduling policy.* Here we work with the simple framework consisting of a single processor. The scheduling policy we shall assume is *preemptive EDF (earliest-deadline-first).* At any given time, the task that is executing on the processor is the one with earliest relative deadline among all the tasks currently in the ready queue. Whenever a fresh task τ' arrives, if its relative deadline is less than the (current) relative deadline of the currently executing task τ, then τ is preempted, placed back in the ready queue (at the head of the queue actually) and τ' will start executing. Thus the state of the task plant will consist of the current location, the values of the clocks associated with the task plant and the state of the ready queue. This motivates the following definition.

A *ready queue* η over Υ is a finite sequence $\sigma_1 \ldots \sigma_n$ where, for $i = 1, \ldots, n$, we have $\sigma_i = \langle \tau_i, c_i, d_i \rangle$, with $\tau_i \in \Upsilon$, $c_i, d_i \in \mathbb{R}_{\geq 0}$, $c_i \leq C(\tau_i)$, and $d_i \leq D(\tau_i)$. Intuitively, σ_i is a *task instance* at position i of type τ_i with *remaining* computation time c_i and *remaining* relative deadline d_i. We assume the convention that σ_1 is the head of the queue. The *empty queue* is denoted ε.

The order in which the task instances appear in the queue reflects the order in which these task instances will be scheduled. If the queue is non-empty, the task instance at the head of the queue is currently supposed to be executing. When time passes, both the remaining computation time (c_i) and the relative deadline (d_i) of the task at the head of the queue will decrease while just the relative deadlines of the remaining tasks will decrease. When the task at the head of the queue finishes, it will leave the ready queue and the task behind it will be promoted to the head of the queue and will start executing. A fresh task instance, when admitted will be inserted into the ready queue at the appropriate slot as dictated by the relative deadline of this instance and the current relative deadlines of the task instances in the ready queue.

The way in which the state of the ready queue changes due to passage of time (and the execution of tasks) is modelled by the function Comp. This function takes a queue η and a time duration t as inputs and returns a queue η' resulting from computing the tasks in η for t time units on the processor and is defined as follows:

- Firstly, $\mathsf{Comp}(\varepsilon, t) = \varepsilon$.
- Secondly suppose $\eta = \sigma_1 \dots \sigma_n$ where $\sigma_i = \langle \tau_i, c_i, d_i \rangle$ for $1 \leq i \leq n$. If $t < c_1$, then $\mathsf{Comp}(\eta, t) = \eta'$, where $\eta' = \sigma'_1 \sigma'_2 \dots \sigma'_n$ with $\sigma'_1 = \langle \tau_1, c_1 - t, d_1 - t \rangle$ and $\sigma'_i = \langle \tau_i, c_i, d_i - t \rangle$ for $1 < i \leq n$.

  If $t \geq c_1$, then $\mathsf{Comp}(\eta, t) = \mathsf{Comp}(\eta', t - c_1)$ where $\eta' = \sigma'_1 \sigma'_2 \dots \sigma'_{n-1}$ with $\sigma'_i = \langle \tau_{i+1}, c_{i+1}, d_{i+1} - c_1 \rangle$ for $1 \leq i \leq n - 1$.

The way in which a task that has just been admitted is inserted into the ready queue is captured by the function Sch which takes as inputs a ready queue η and a task instance J and returns a ready queue η'.

Let $\eta = \langle \tau_1, c_1, d_1 \rangle \dots \langle \tau_n, c_n, d_n \rangle$ be a ready queue, where $d_1 \leq \cdots \leq d_n$, and let $J = \langle \tau, C(\tau), D(\tau) \rangle$ be a task instance. Then $\mathsf{Sch}(\eta, J) = \eta'$, where $\eta' = \langle \tau_1, c_1, d_1 \rangle \dots \langle \tau_{k-1}, c_{k-1}, d_{k-1} \rangle \langle \tau, C(\tau), D(\tau) \rangle \langle \tau_k, c_k, d_k \rangle \dots \langle \tau_n, c_n, d_n \rangle$

with k being the least index such that $D(\tau) < d_k$. Clearly this reflects the preemptive EDF policy.

A queue $\eta = \langle \tau_1, c_1, d_1 \rangle \ldots \langle \tau_n, c_n, d_n \rangle$ is *schedulable* iff $\sum_{i=1}^{m} c_i \leq d_m$ for $1 \leq m \leq n$. That is, assuming that no fresh task instances are inserted into the ready queue, each task instance currently residing in the ready queue can be scheduled (according to EDF) and run to completion before its current deadline expires. In what follows, *all* non-schedulable queues will be identified with the single designated (non-schedulable) queue named Err. The functions Comp and Sch are extended in the obvious way to reflect this convention.

4.3.4. The Task Plant Semantics.

The semantics of the task plant $\mathcal{A}$ is defined by a transition system $TS_{\mathcal{A}} = (RC_{\mathcal{A}}, \langle q_0, V_0, \varepsilon, \sharp \rangle, ACT_{\mathcal{A}}, \Longrightarrow_{\mathcal{A}})$, where

$$ACT_{\mathcal{A}} = (\mathbb{R}_+ \times \Upsilon) \bigcup \{0, 1\} \bigcup \{\mathsf{err}\} \ .$$

And $RC_{\mathcal{A}}$, the set of *(reachable) configurations*, and $\Longrightarrow_{\mathcal{A}}$ are the least sets satisfying the following:

- Firstly, $\langle q_{in}, V_{in}, \varepsilon, \sharp \rangle \in RC_{\mathcal{A}}$.
- Secondly, suppose $\langle q, V, \eta, \sharp \rangle \in RC_{\mathcal{A}}$ and $q \xrightarrow{\varphi, \tau, Y}_e q'$. If there exists $t \in \mathbb{R}_+$ such that $V + t \models \varphi$, then $\langle q', V', \eta', J \rangle \in RC_{\mathcal{A}}$ and $\langle q, V, \eta, \sharp \rangle \overset{t,\tau}{\Longrightarrow}_{\mathcal{A}} \langle q', V', \eta', J \rangle$, where $V' = (V + t)[Y := 0]$, $\eta' = \mathsf{Comp}(\eta, t)$, and $J = \langle \tau, C(\tau), D(\tau) \rangle$.
- Thirdly, suppose $\langle q, V, \eta, J \rangle \in RC_{\mathcal{A}}$ and $q \xrightarrow{0}_s q'$. Then $\langle q', V, \eta, \sharp \rangle$ is in $RC_{\mathcal{A}}$ and $\langle q, V, \eta, J \rangle \overset{0}{\Longrightarrow}_{\mathcal{A}} \langle q', V, \eta, \sharp \rangle$.
- Fourth, suppose $\langle q, V, \eta, J \rangle \in RC_{\mathcal{A}}$ and $q \xrightarrow{1}_s q'$. If $\mathsf{Sch}(\eta, J) = \eta' \neq \mathsf{Err}$, then $\langle q', V, \eta', \sharp \rangle \in RC_{\mathcal{A}}$ and $\langle q, V, \eta, J \rangle \overset{1}{\Longrightarrow}_{\mathcal{A}} \langle q', V, \eta', \sharp \rangle$. If $\mathsf{Sch}(\eta, J) = \mathsf{Err}$, then $\mathsf{Err} \in RC_{\mathcal{A}}$ and $\langle q, V, \eta, J \rangle \overset{1}{\Longrightarrow}_{\mathcal{A}} \mathsf{Err}$.
- Finally, $\mathsf{Err} \overset{\mathsf{err}}{\Longrightarrow}_{\mathcal{A}} \mathsf{Err}$.

Configurations of the form $\langle q, V, \eta, \sharp \rangle$ are called *environment configurations* where $\sharp$ is a special symbol, while those of the form $\langle q, V, \eta, J \rangle$ are called *system configurations.* Note that Err is neither an environment configuration nor a system configuration. Intuitively, $TS_{\mathcal{A}}$ is the game graph of a real-time system where the environment triggers tasks in an uncontrollable manner, while the system admits or rejects them in a controllable way. Note that we require environment moves to consume a nonzero amount of time and system moves to consume *no time.*

In what follows, we shall often omit the subscript $\mathcal{A}$. For a configuration σ, we define $\mathrm{Act}(\sigma) = \{a \in ACT \mid \exists \sigma'.\ \sigma \overset{a}{\Longrightarrow} \sigma'\}$. That is, $\mathrm{Act}(\sigma)$ is the set of possible actions that can be taken at configuration σ. Note that $\mathrm{Act}(\sigma) \subseteq \mathbb{R}_+ \times \Upsilon$ if σ is an environment configuration, $\mathrm{Act}(\sigma) = \{0, 1\}$ if σ is a system configuration σ and $\mathrm{Act}(\mathsf{Err}) = \{\mathsf{err}\}$.

Without loss of generality, we shall assume that the task plant has no dead configuration, that is, $\mathrm{Act}(\sigma) \neq \emptyset$ for $\sigma \in RC$. Given the restrictions on the structure of the task plant transitions, we can uniquely recover a sequence of task plant transitions from each sequence of transitions of TS with the help of the projection operator $\mathrm{Prj}_{\mathcal{A}}$. Let $\delta = \langle \langle q, V, \eta, \sharp \rangle, (t, \tau), \langle q', V', \eta', J \rangle \rangle$ be an environment transition in TS. Then clearly, there is a unique transition $\beta = (q \xrightarrow{\varphi, \tau, Y}_e q')$ in $\mathcal{A}$ such that $V + t'$ satisfies I_q for $t' \leq t$, $V + t$ satisfies φ, $V' = (V + t)[Y := 0]$. In this case, we define $\mathrm{Prj}_{\mathcal{A}}(\delta) = \beta$. For a system move $\delta = \langle \langle q, V, \eta, J \rangle, m, \langle q', V', \eta', \sharp \rangle \rangle$ in TS, we define $\mathrm{Prj}_{\mathcal{A}}(\delta) = (q \xrightarrow{m}_s q')$. For the special move $\delta = \langle \mathsf{Err}, \mathsf{err}, \mathsf{Err} \rangle$, we define $\mathrm{Prj}_{\mathcal{A}}(\delta) = \delta$. Using the operator $\mathrm{Prj}_{\mathcal{A}}$, it is clear that we can uniquely associate a sequence of transitions of the task plant with each transition sequence of TS (modulo the handling of the err moves).

4.4. The Admission Controller Synthesis Problem

Here we define the admission controller synthesis problem formally.

Let $\mathcal{A}$ be the task plant as fixed in the preceding section. We say $\mathcal{A}$ is said to be *schedulable* iff $\mathsf{Err} \notin RC_{\mathcal{A}}$. That is, no task will ever miss its deadline, even if we never reject any task. It is proved in [24]—in a related closed system setting—that one can effectively determine if $\mathcal{A}$ is schedulable, under the *preemptive EDF policy* for a single processor.

Here we consider task plants which may not be schedulable and study the problem of designing an admission policy under which the restricted behaviour of the plant becomes schedulable. Clearly, one can trivially achieve schedulability by constantly rejecting all arrived tasks. To rule out this, we shall assume that we are also given a liveness specification. Here we shall assume that LTL (linear time temporal logic) is the chosen specification language. In order to formulate this, we fix a finite set of atomic propositions AP and a labelling function $\lambda : \Delta_{\mathcal{A}} \to 2^{AP}$, where $\Delta_{\mathcal{A}}$ consists of all transitions in $\longrightarrow_e$ and all transitions in $\longrightarrow_s$. The target states of both the 0 and 1 transitions coming out of a system state are the same. To capture liveness properties we need to get the admission decisions made by the system during a run. Hence we assign atomic propositions to transitions than to states. From now on we shall assume our task plant model to be augmented with AP and a labelling function λ as specified above.

Let Δ_{TS} be the set of transitions of TS, that is, the set of transitions in $\Longrightarrow_{\mathcal{A}}$. Using the operator Prj we extend λ to Δ_{TS}. By abuse of notation this extension will also be denoted as λ and it is given by: $\lambda(\delta) = \lambda(\mathrm{Prj}(\delta))$ for every $\delta \in \Delta_{TS}$. By convention, we will set $\lambda(\langle \mathsf{Err}, \mathsf{err}, \mathsf{Err} \rangle) = \emptyset$. Let $\chi = \beta_0\, \beta_1 \ldots$ be an infinite sequence of transitions of TS. Let ψ be an LTL formula over AP. We shall say χ is a model of ψ iff the infinite sequence $\lambda(\beta_0)\, \lambda(\beta_1)\, \ldots$ over 2^{AP} is a model of ψ.

We are now ready to define the notion of an admission strategy. Define the set of *(finite) runs* of TS, denoted $\mathit{Runs}(TS)$, to be the least subset of finite sequences over Δ_{TS} satisfying:

- $\varepsilon \in \mathit{Runs}(TS)$ and the *end state* of ε is $\langle q_{in}, V_{in}, \varepsilon, \sharp \rangle$.
- Suppose $\rho \in \mathit{Runs}(TS)$ and the *end state* of ρ is σ. Suppose further that there exists $\delta = \langle \sigma, a, \sigma' \rangle$ in Δ_{TS}. Then $\rho\,\delta \in \mathit{Runs}(TS)$ and the *end state* of $\rho\,\delta$ is σ'.

In what follows, we denote the end state of a run ρ in $\mathit{Runs}(TS)$ by $\mathrm{end}(\rho)$.

An *admission strategy* f for the task plant $\mathcal{A}$ is a function f from $\mathit{Runs}(TS)$ into 2^{ACT} such that for every ρ in $\mathit{Runs}(TS)$, the following conditions hold (where $\sigma = \mathrm{end}(\rho)$):

- If σ is a system configuration, then $f(\rho)$ is nonempty subset of $\mathrm{Act}(\sigma)$ (note that $\mathrm{Act}(\sigma) = \{0, 1\}$ in this case).
- If σ is an environment configuration, then $f(\rho) = \mathrm{Act}(\sigma)$.
- If $\sigma = \mathsf{Err}$, then $f(\rho) = \emptyset$.

Thus an admission strategy recommends a set of moves at the end state of each run. The moves recommended are a subset of the moves enabled at the end state and at least one move is recommended. In this sense an admission strategy is, by definition, *non-blocking*. Note that an admission strategy does not restrict the moves of the environment in any way.

Let f be an admission strategy for $\mathcal{A}$. We define the set of runs in $\mathit{Runs}(TS)$ being *according to* f, denoted $\mathit{Runs}(f)$, inductively as follows:

- ε is in $\mathit{Runs}(f)$.
- Suppose ρ in $\mathit{Runs}(TS)$ is in $\mathit{Runs}(f)$ and $\sigma = \mathrm{end}(\rho)$. Suppose further that $a \in f(\rho)$ and $\delta = \langle \sigma, a, \sigma' \rangle$ is in Δ_{TS}. Then $\rho\,\delta$ is in $\mathit{Runs}(f)$.

Infinite runs of TS and the notion of an infinite run of TS being according to an admission strategy are defined in the usual way. Namely, an infinite run of TS is an infinite sequence ρ over Δ_{TS} such that every finite prefix of ρ is in $Runs(TS)$. For an infinite run ρ of TS, we say ρ is according to the admission strategy f iff every finite prefix of ρ is according to f.

We say the admission strategy f for $\mathcal{A}$ is *safe* iff for any run ρ in $Runs(f)$, the end state of ρ is not Err. Given an LTL or QPLTL formula ψ, we will say that the admission strategy f is *ψ-winning* iff the following conditions are satisfied: firstly, f is safe; secondly, if $\rho = \delta_0\,\delta_1\,\ldots$ is any infinite run of TS according to f, then ρ is a model of ψ. By a *ψ-controller* for $\mathcal{A}$, we shall mean a ψ-winning admission strategy.

We can now state the *admission controller synthesis* problem formally: Given a task plant $\mathcal{A}$ and an LTL or QPLTL formula ψ, does there exist an admission controller.

4.5. Decidability Results

Our result is that the admission controller synthesis problem is decidable and moreover, if an admission controller exists, then one can effectively construct a finite state admission controller in the form of a (finite) timed automaton.

We begin with QoS specifications in LTL.

Theorem 4.2.

(i) *Given a task plant $\mathcal{A}$ and an LTL formula ψ. we can effectively determine whether there exists a ψ-controller for $\mathcal{A}$.*

(ii) *Moreover, whenever such a ψ-controller exists, one can effectively construct a ψ-controller in the form of a finite timed automaton.*

The proof of theorem 4.2(i) is based on two ideas. Firstly, as shown in [24], we can associate clock variables with the ready queue and capture its dynamics with the help of a mildly extended timed automaton model that uses only the ready queue's clock variables. We review this construction in section 4.5.1. Secondly, we prove in section 4.5.2 the important lemma that if there is a winning admission strategy at all then there is *region-respecting* one which respects the regional equivalence induced by the plant's and the ready queue's clock variables. Consequently, it suffices to determine whether or not there exists a region-respecting winning admission strategy. And this allows us to work with the regional version of *TS* which will be a *finite* transition system.

In section 4.5.3, we reduce the problem of determining the existence of a winning admission strategy to checking the nonemptiness of the language of a non-deterministic Rabin tree automaton $\mathcal{R}_{ctrl}$ that we can effectively construct. The tree automaton $\mathcal{R}_{ctrl}$ will run over $\{\top, \bot\}$-labelled trees whose underlying tree will be the computation tree induced by the regional version of $\mathcal{A}$. Such a tree will be accepted by $\mathcal{R}_{ctrl}$ iff the $\{\top, \bot\}$-labelling represents a winning strategy. This will settle theorem 4.2(i). We then analyze the complexity of the decision procedure in theorem 4.2(i).

In section 4.5.4, we discuss that the proof of theorem 4.2(i) can also be handled using zones, instead of regions.

To prove theorem 4.2(ii), we shall make use of Rabin's tree theorem [63]. It implies that in case the set of $\{\top, \bot\}$-labelled trees accepted by $\mathcal{R}_{ctrl}$ is nonempty, then $\mathcal{R}_{ctrl}$ in fact accepts a *regular* $\{\top, \bot\}$-labelled tree. In section 4.5.5, we show that this regular tree can be effectively computed, represented as a finite structure and this structure can be naturally viewed as a timed automaton. This timed automaton will constitute the winning admission strategy we seek.

Lastly, in section 4.5.6, we show that the proof of theorem 4.2 can be easily adapted to handle QPLTL formulae.

4.5.1. Timed Automaton for the Ready Queue.

Here we briefly summarize the construction in [24] of a timed automaton extended with subtraction $\mathcal{A}_{qu}$ which describes the dynamics of the ready queue under the preemptive EDF scheduling policy in a uniprocessor setting.

A timed automaton extended with subtraction is just an ordinary timed automaton, in which a *constant integer* value may be subtracted from a clock value during a transition. However such subtractions will not be allowed in arbitrary contexts. They will be used in a manner which ensures that no clock value becomes negative via subtraction. Moreover, a clock value will be subtracted from only when its value is below the maximum constant associated with it. Due to these two properties, the region construction of ordinary timed automata can also be applied to $\mathcal{A}_{qu}$.

For illustration, we show a timed automaton in in figure 4.2. We display its clock regions in figure 4.3 and its region automaton in figure 4.4

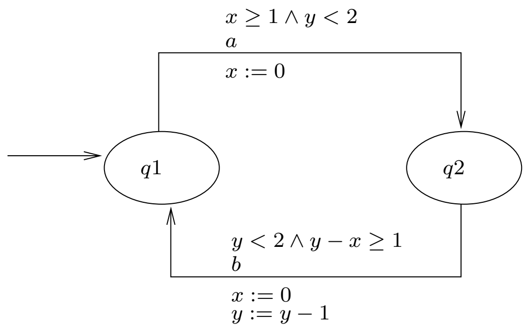


FIGURE 4.2. A timed automaton extended with subtraction

Observe that a schedulable ready queue contains at most $\lceil D(\tau)/C(\tau) \rceil$ instances of each task type $\tau \in \Upsilon$. Let (τ, j) denote the j-th instance of τ, for $j = 1, \ldots, \lceil D(\tau)/C(\tau) \rceil$. The function stat, to be associated with the locations of $\mathcal{A}$ will give the status of each potential instance (τ, j).

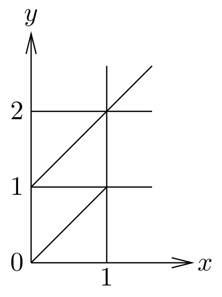


FIGURE 4.3. Clock regions

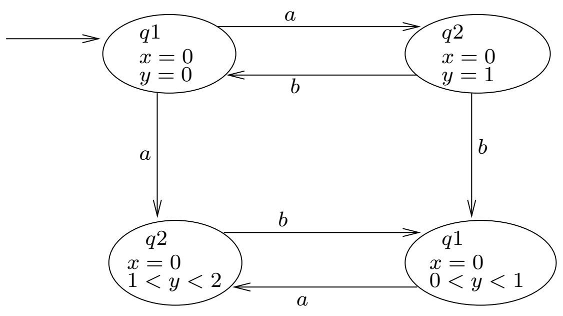


FIGURE 4.4. The region automaton

The status of an instance can be out (not in the ready queue), in (released and *in* the ready queue, but has not started execution), exec (being *executed* by the processor) or pre (started execution but has been *preempted* by another instance). The set of locations of $\mathcal{A}$ consist of these status functions and a special location Err_{qu} designated to represent all non-schedulable ready queues. The initial location is the status function representing the empty queue ε, denoted ε_{qu}.

The event alphabet is $\Upsilon \cup \{\mathsf{err}_{qu}\}$. An event $\tau \in \Upsilon$ signifies the arrival of an instance of τ. The event err_{qu} is a dummy one which is used only for the special location Err_{qu}.

For each instance (τ, j), two clocks $xc(\tau, j)$ and $xd(\tau, j)$ are used to keep track of its remaining computation time and remaining relative deadline. More precisely, the remaining computation time will be $C(\tau) - xc(\tau, j)$ and the remaining relative deadline will be $D(\tau) - xd(\tau, j)$. The set X_{qu} of all such clocks $xc(\tau, j)$, $xd(\tau, j)$, where $\tau \in \Upsilon$, $j \leq \lceil D(\tau)/C(\tau) \rceil$, is the clock set of $\mathcal{A}_{qu}$. These clocks evolve as follows:

- $xd(\tau, j)$ is reset to zero when (τ, j) is released. $xc(\tau, j)$ is reset to zero when it starts execution.
- When (τ, j) finishes execution (and exits the ready queue), we subtract $C(\tau)$ from every $xc(\sigma, k)$ whose status is pre.

For illustration, we display in figure 4.5 a fragment of the timed automaton $\mathcal{A}_{qu}$ for the task types τ_a, τ_b associated with the task plant in figure 4.1. To reduce clutter, we have omitted those task instances whose statuses are out. A transition labelled with action τ_a (respectively, τ_b) indicates the arrival of a new task instance of τ_a (respectively, τ_b). The action labels of other transitions are immaterial and hence omitted.

Let $TS_{\mathcal{A}_{qu}}$ (TS_{qu} for short) be the transition system associated with $\mathcal{A}_{qu}$. From [24], we have the following.

Proposition 4.3 ([24]). *There exists a 1-1 correspondence Ω between ready queues and configurations in TS_{qu} which satisfies:*

- $\Omega(\varepsilon) = (\varepsilon_{qu}, V^{qu}_{in})$, *where* $V^{qu}_{in}(x) = 0$ *for each* $x \in X_{qu}$.
- *Suppose* $\Omega(\eta_1) = (q_1, V_1), \Omega(\eta_2) = (q_2, V_2)$. *Let* $t \in \mathbb{R}_+$ *and* $J = \langle \tau, C(\tau), D(\tau) \rangle$ *be a task instance. Then* $\eta_2 = \mathsf{Comp}(\eta_1, t)$ *iff* $\langle (q_1, V_1), t, (q_2, V_2) \rangle$ *is a transition of* TS_{qu}. *Further,* $\eta_2 = \mathsf{Sch}(\eta_1, J)$ *iff* $\langle (q_1, V_1), \tau, (q_2, V_2) \rangle$ *is a transition of* TS_{qu}.
- $\Omega(\mathsf{Err}) = (\mathsf{Err}_{qu}, V^{qu}_{in})$.

Hence, from now on, we shall replace the ready queue component in configurations of TS by the corresponding configuration in TS_{qu}. To be specific, the configuration $\langle q, V, \eta, \kappa \rangle$ of TS will now be represented as $\langle q, V, q', V', \kappa \rangle$ where $\Omega(\eta) = (q', V')$.

We define the *regional equivalence relation* $\sim$ to be the least equivalence relation on RC satisfying: Suppose $\sigma_1 = \langle q1, V1, q'_1, V'_1, \kappa_1 \rangle$, $\sigma_2 =$

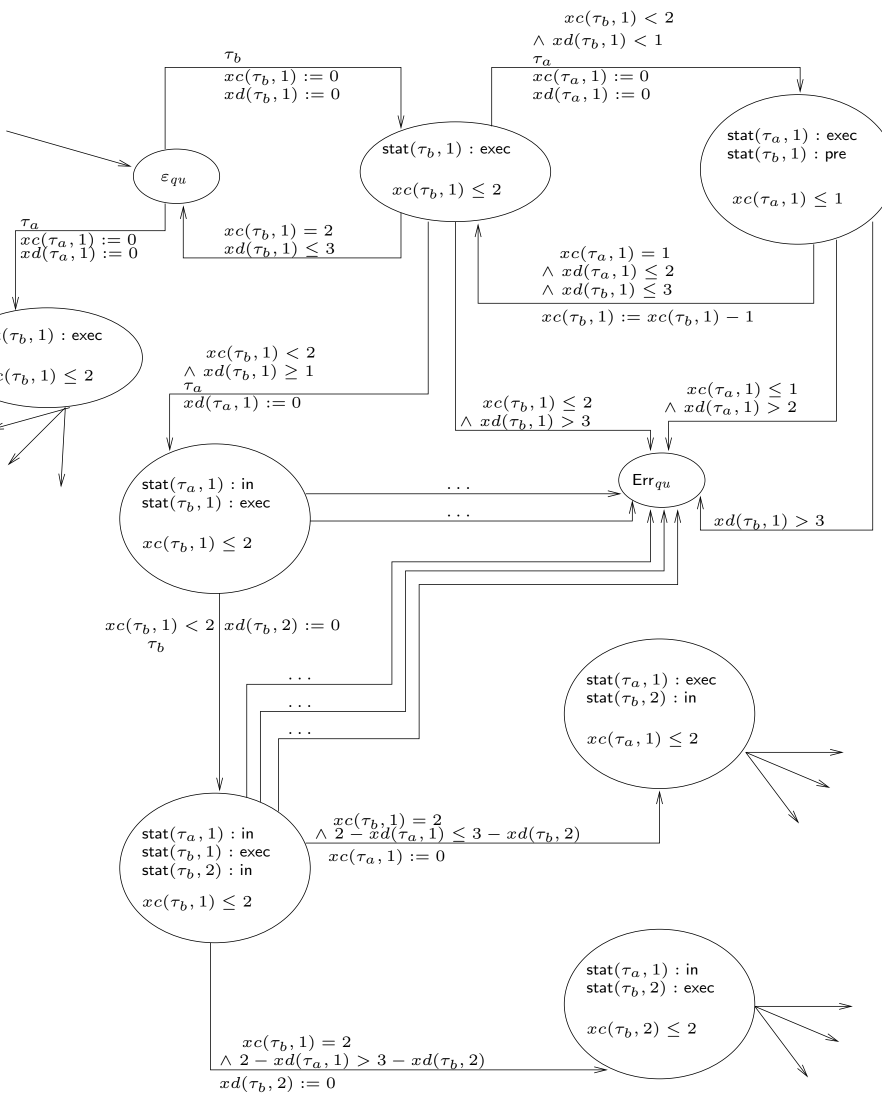


FIGURE 4.5. The timed automaton $\mathcal{A}_{qu}$

$\langle q_2, V_2, q_2', V_2', \kappa_2\rangle$ are configurations in RC. Then $\sigma_1 \sim \sigma_2$ iff $q_1 = q_2$, $q_1' = q_2'$, $\kappa_1 = \kappa_2$, and the clock valuations $V_1 \cdot V_1'$ and $V_2 \cdot V_2'$ belong to the same clock region with respect to the clocks in $X \cup X_{qu}$, where $V \cdot V'$ denotes the clock valuation over $X \cup X_{qu}$ which agrees with V on clocks of X and agrees with V' on clocks of X_{qu}. In particular, Err is region-equivalent only to itself (if Err is in RC).

Using the arguments developed in [24], it is easy to prove the following result.

Proposition 4.4 ([24]). *Suppose σ_1, σ_2 are configurations in RC with $\sigma_1 \sim \sigma_2$. Then we have the following:*

(i) *Suppose σ_1 (hence also σ_2) is an environment configuration. If $\sigma_1 \overset{t,\tau}{\Longrightarrow} \sigma_3$, then there exists $t' \in \mathbb{R}_+$ such that $\sigma_2 \overset{t',\tau}{\Longrightarrow} \sigma_4$* and *$\sigma_3 \sim \sigma_4$.*

(ii) *Suppose σ_1 (hence also σ_2) is a system configuration. If $\sigma_1 \overset{m}{\Longrightarrow} \sigma_3$ and $\sigma_2 \overset{m'}{\Longrightarrow} \sigma_4$, where $m, m' \in \{0,1\}$, then $m = m'$ iff $\sigma_3 \sim \sigma_4$.*

We are now ready to define $RA_{\mathcal{A}}$, the *regional version* of $\mathcal{A}$. A *region* of $\mathcal{A}$ is an equivalence class of $\sim$ on RC. For $\sigma \in RC$, we let $[\sigma]_\sim$ denote the region containing σ and often just write $[\sigma]$. We define $RA = (RG, [\langle q_{in}, V_{in}, \varepsilon_{qu}, V_{in}^{qu}, \sharp\rangle], ACT_{RA}, \rightsquigarrow)$ where RG is the set of regions and $ACT_{RA} = \Upsilon \cup \{0,1\} \cup \{\mathsf{err}\}$. Further, $\rightsquigarrow$ is least set which satisfies the following:

- Suppose $[\sigma_1]$, $[\sigma_2]$ are in RG where σ_1 is an environment configuration and σ_2 is a system configuration. Let τ be in Υ. Then $[\sigma_1] \overset{\tau}{\Longrightarrow} [\sigma_2]$ iff there exist σ_1' in $[\sigma_1]$ and σ_2' in $[\sigma_2]$ and t in $\mathbb{R}_+$ such that $\sigma_1' \overset{t,\tau}{\Longrightarrow} \sigma_2'$.
- Suppose $[\sigma_1]$, $[\sigma_2]$ are in RG where σ_1 is a system configuration and σ_2 is an environment configuration. Let $m \in \{0,1\}$. Then $[\sigma_1] \overset{m}{\Longrightarrow} [\sigma_2]$ iff there exists σ_1' in $[\sigma_1]$, σ_2' in $[\sigma_2]$ such that $\sigma_1' \overset{m}{\rightsquigarrow} \sigma_2'$.
- If Err is in RC, then $[\mathsf{Err}] \overset{\mathsf{err}}{\rightsquigarrow} [\mathsf{Err}]$.

It follows that RG is finite. It is also easy to see that the construction of RA is effective.

4.5.2. Region-Respecting Strategies.

In order to prove theorem 4.2(i), we first show a crucial lemma which states that if there exists a ψ-winning admission strategy for $\mathcal{A}$, then there exists a *region-respecting* ψ-winning admission strategy for $\mathcal{A}$. This lemma will then allow us to work with the regional version of the task plant and consider only region-respecting admission strategies. Intuitively, we say an admission strategy is region-respecting iff it does not distinguish two runs in $\mathit{Runs}(\mathit{TS})$ that pass through the same sequence of regions.

Formally, we define the notion of an admission strategy being region-respecting as follows. Abusing notation, we first extend the region equivalence relation $\sim$ on RC to $\mathit{Runs}(\mathit{TS})$ as follows. Suppose ρ, $\hat{\rho}$ are runs in $\mathit{Runs}(\mathit{TS})$, where

$$\rho = \langle \sigma_0, a_0, \sigma'_0 \rangle \, \langle \sigma_1, a_1, \sigma'_1 \rangle \, \ldots \, \langle \sigma_n, a_n, \sigma'_n \rangle \ ,$$

$$\hat{\rho} = \langle \hat{\sigma}_0, \hat{a}_0, \hat{\sigma}'_0 \rangle \, \langle \hat{\sigma}_1, \hat{a}_1, \hat{\sigma}'_1 \rangle \, \ldots \, \langle \hat{\sigma}_n, \hat{a}_n, \hat{\sigma}'_n \rangle \ .$$

Then we say $\rho \sim \rho'$ iff for every $i = 1, 2, \ldots, n$, it is the case that $\sigma_i \sim \hat{\sigma}'_i$, $a_i = \hat{a}_i$, and $\sigma'_i \sim \hat{\sigma}'_i$. Now we say the admission strategy f is *region-respecting* iff for every ρ, $\hat{\rho}$ in $\mathit{Runs}(\mathit{TS})$, if ρ, $\hat{\rho}$ are in $\mathit{Runs}(f)$ and $\rho \sim \hat{\rho}$, then $f(\rho) = f(\hat{\rho})$.

The following result is crucial for establishing theorem 4.2.

Lemma 4.5. *Let $\mathcal{A}$ be the task plant as fixed above. Suppose ψ is an LTL specification. Then there exists a ψ-winning admission strategy for $\mathcal{A}$ iff there exists a region-respecting ψ-winning admission strategy for $\mathcal{A}$.*

Proof. Suppose f is a ψ-winning strategy for $\mathcal{A}$. We inductively construct a prefix-closed subset REP_f of $\mathit{Runs}(f)$ such that REP_f is "representative" of $\mathit{Runs}(f)$.

Firstly $\varepsilon \in \mathit{REP}_f$. We recall that $\mathrm{end}(\varepsilon)$ is $\langle q_{in}, V_{in}, \varepsilon_{qu}, V_{in}^{qu}, \sharp \rangle$.

Next suppose $\rho \in \mathit{REP}_f$ where $\mathrm{end}(\rho) = \sigma$ is an environment configuration. For $[\hat{\sigma}]$ in RG, we will say that $[\hat{\sigma}]$ is a *region successor* of σ iff there exists $\hat{\sigma}'$ in $[\hat{\sigma}]$ and (t, τ) in $\mathbb{R}_+ \times \Upsilon$ such that $\sigma \overset{t,\tau}{\Longrightarrow} \hat{\sigma}'$. Let $[\hat{\sigma}_1], [\hat{\sigma}_2], \ldots, [\hat{\sigma}_k]$ be the set of region successors of σ. For each $i = 1, 2, \ldots, k$, we pick (t_i, τ_i) such that $\hat{\sigma}'_i$ is in $[\hat{\sigma}_i]$, where $\hat{\sigma}'_i$ is the *unique* configuration in RC satisfying $\sigma \overset{t_i,\tau_i}{\Longrightarrow} \hat{\sigma}'_i$. We then let $\rho\,\delta_i$ be in REP_f, where $\delta_i = \langle \sigma, (t_i, \tau_i), \hat{\sigma}'_i \rangle$, for every $i = 1, 2, \ldots, k$.

We argue that the choices of t_i, $i = 1, 2, \ldots, k$, can be made effective, and hence one need *not* appeal to the axiom of choice ([34]) here. Now fix $i \in \{1, 2, \ldots, k\}$. Clearly, the task type τ_i exists and is unique. Let T_i be the set of reals given by: t in $\mathbb{R}_+$ is in T_i iff σ'_i is in $[\hat{\sigma}_i]$ where σ'_i is the unique configuration in RC with $\sigma \overset{t,\tau_i}{\Longrightarrow} \sigma'_i$.

Let $\hat{\sigma}_i = \langle \hat{q}_i, \widehat{V}_i, \hat{q}'_i, \widehat{V}'_i, \hat{\kappa}_i \rangle$. We note that the region $[\hat{\sigma}_i]$ is in fact represented as a tuple consisting of q_i, q'_i, κ_i and the clock region containing $\widehat{V} \cdot \widehat{V}'$ that is in turn represented as a conjunction of clock constraints of the forms

$$x = c,\ c - 1 < x < c,\ x > c_x,\ x - y = c,\ c - 1 < x - y < c,\ x - y \prec c',$$

where x, y in $X \cup X_{qu}$ and $\prec$ is in $\{<, \leq, >, \geq\}$. And c_x is the maximum constant which appears in basic constraints of $\mathcal{A}$ and $\mathcal{A}_{qu}$; $c \in \mathbb{N}$ with $c \leq c_x$; and $x - y \prec c'$ is a difference constraint which appears in basic constraints of $\mathcal{A}$ and $\mathcal{A}_{qu}$. It follows that T_i is actually an interval, namely, there exist r, r' in $\mathbb{R}_{\geq 0}$ (respectively, $r \in \mathbb{R}_{\geq 0}$ such that $t \in T_i$ iff $r \prec t \prec r'$ (respectively, $r \prec t$), where $\prec$ is in $\{<, \leq\}$. Hence we pick t_i to be $\frac{1}{2}(r + r')$ (respectively, $r + 1$).

Finally, suppose $\rho \in REP_f$ where $\mathrm{end}(\rho) = \sigma$ is a system configuration. Then for each $m \in f(\rho)$, we let $\rho\,\delta$ be in REP_f, where $\delta = \langle \sigma, m, \sigma' \rangle$ with σ' being the unique (environment) configuration in RC with $\sigma \overset{m}{\Longrightarrow} \sigma'$.

It is clear that REP_f is indeed a prefix-closed subset of $Runs(f)$.

Now we construct the region-respecting admission strategy $\hat{f}$ from REP_f as follows. For ρ in $Runs(TS)$, if there exists ρ' in REP_f such that $\rho' \sim \rho$, then $\hat{f}(\rho) = f(\rho')$; otherwise $\hat{f}(\rho) = \emptyset$. It is now routine to show that $\hat{f}$ is in fact ψ-winning.

□

4.5.3. Decidability for LTL Specifications.

With lemma 4.5, we are now ready to prove theorem 4.2(i). It follows from lemma 4.5, to check if a ψ-winning admission strategy exists, it suffices to determine if a region-respecting ψ-winning admission strategy exists. We now show that the latter can be done effectively and hence establish theorem 4.2(i).

In what follows, we fix a two-letter alphabet $\{\top, \bot\}$, where $\top$, $\bot$ are special symbols. We show that region-respecting admission strategies can be represented as $\{\top, \bot\}$-labelled trees, whose underlying tree is basically the computation tree induced by RA. And one can effectively construct a non-deterministic Rabin tree automaton which accepts the set of $\{\top, \bot\}$-labelled trees representing region-respecting ψ-winning admission strategies.

We set Δ_{RA} to be the set of transitions of RA, that is, the set of transitions in $\rightsquigarrow$. For a region $[\sigma]$ in RG, we define $\mathit{Tran}([\sigma])$ to be the subset of Δ_{RA} given by: $\langle [\hat{\sigma}], a, [\hat{\sigma}'] \rangle$ in Δ_{RA} is in $\mathit{Tran}([\sigma])$ iff $[\hat{\sigma}] = [\sigma]$. Now we construct the Δ_{RA}-tree $\mathcal{T}$ inductively as follows:

- Firstly, $\varepsilon \in \mathcal{T}$ and the end state of ε is $[\langle q_{in}, V_{in}, \varepsilon_{qu}, V_{in}^{qu}, \sharp \rangle]$.
- Secondly, suppose $\rho \in \mathcal{T}$ and the end state of ρ is $[\sigma]$ (in RG). Then for each $\langle [\sigma], a, [\sigma'] \rangle$ in $\mathit{Tran}([\sigma])$, we let $\rho\,\delta$ be in $\mathcal{T}$, where $\delta = \langle [\sigma], a, [\sigma'] \rangle$, and set the end state of $\rho\,\delta$ to be $[\sigma']$.

By abuse of notation, we will also denote the end state of a node ρ in $\mathcal{T}$ as end(ρ). Clearly $\mathcal{T}$ is a Δ_{RA}-tree and for every ρ in $\mathcal{T}$, $\mathit{Succ}_{\mathcal{T}}(\rho) \neq \emptyset$.

Let $(\mathcal{T}, \xi)$ be a $\{\top, \bot\}$-labelled Δ_{RA}-tree, where $\eta : \mathcal{T} \to \{\top, \bot\}$ is a labelling function. We say $(\mathcal{T}, \xi)$ is an *admission strategy tree* iff the following conditions hold:

- $\xi(\varepsilon) = \top$.

- Suppose ρ is in $\mathcal{T}$ where $\mathrm{end}(\rho) = [\sigma]$ with σ being an environment configuration in RC. Then for any δ in $\mathit{Tran}([\sigma])$, we have $\xi(\rho\,\delta) = \top$.
- Suppose ρ is in $\mathcal{T}$ where $\mathrm{end}(\rho) = [\sigma]$ with σ being a system configuration in RC. Then for there exists δ in $\mathit{Tran}([\sigma])$ such that $\xi(\rho\,\delta) = \top$.
- Suppose ρ is in $\mathcal{T}$ where $\mathrm{end}(\rho) = \mathsf{Err}$. Then for any δ in $\mathit{Tran}([\mathsf{Err}])$, we have $\xi(\rho\,\delta) = \bot$.

It is easy to see that there is a 1-1 correspondence between region-respecting admission strategies and admission strategy trees. In fact, if f is a region-respecting admission strategy, then one can define the $\{\top, \bot\}$-labelled Δ_{RA}-tree $(\mathcal{T}, \xi_f)$ as follows: for a node

$$\rho = \langle [\sigma_0], a_0, [\sigma_0'] \rangle \, \langle [\sigma_1], a_1, [\sigma_1'] \rangle \; \ldots \; \langle [\sigma_n], a_n, [\sigma_n'] \rangle$$

in $\mathcal{T}$, we have $\xi_f(\rho) = \top$ if there exists

$$\hat{\rho} = \langle \hat{\sigma}_0, \hat{a}_0, \hat{\sigma}_0' \rangle \, \langle \hat{\sigma}_1, \hat{a}_1, \hat{\sigma}_1' \rangle \; \ldots \; \langle \hat{\sigma}_n, \hat{a}_n, \hat{\sigma}_n' \rangle$$

in $\mathit{Runs}(\mathit{TS})$ such that for $i = 1, 2, \ldots, n$, it is the case that $\hat{\sigma}_i$ is in $[\sigma_i]$ and $\hat{a}_i = a_i$ and $\hat{\sigma}_i'$ is in $[\sigma_i']$. Further, $\eta_f(\rho) = \bot$ otherwise. It is clear that $(\mathcal{T}, \xi_f)$ is an admission strategy tree. On the other hand, suppose $(\mathcal{T}, \xi)$ is an admission strategy tree, then one can define a region-respecting admission strategy f_ξ as follows: for ρ in $\mathcal{T}$ with $\mathrm{end}(\rho) = [\sigma]$, we have:

- If σ is an environment configuration, then $f_\xi(\rho) = \mathrm{Act}(\sigma)$.
- If σ is a system configuration, then $m \in \{0, 1\}$ is in $f_\xi(\rho)$ iff the node $\rho\,\delta$ in $\mathcal{T}$ is labelled $\top$ by ξ, where $\delta = \langle [\sigma], m, [\sigma'] \rangle$ with $[\sigma']$ being the unique region such that $\langle [\sigma], m, [\sigma'] \rangle$ is in Δ_{RA}.
- If $\sigma = \mathsf{Err}$, then $f_\xi(\rho) = \emptyset$.

It is straightforward to verify that f_η is a well-defined admission strategy and is region-respecting.

Now it is routine to show that an admission strategy tree $(\mathcal{T}, \xi)$ represents a region-respecting ψ-winning admission strategy iff $(\mathcal{T}, \xi)$ satisfies the following conditions:

- (safe) For every ρ in $\mathcal{T}$ with $\mathrm{end}(\rho) = [\mathsf{Err}]$, we have $\xi(\rho) = \bot$.
- Suppose $\rho = \langle[\sigma_0], a_0, [\sigma_0']\rangle\,\langle[\sigma_1], a_1, [\sigma_1']\rangle\,\ldots$ is a path in $\mathcal{T}$. If every finite prefix of ρ is labelled $\top$ by ξ, then for every infinite run $\hat{\rho} = \langle\hat{\sigma}_0, \hat{a}_0, \hat{\sigma}_0'\rangle\,\langle\hat{\sigma}_1, \hat{a}_1, \hat{\sigma}_1'\rangle\,\ldots$ of TS, where for $i = 0, 1, \ldots$, $\hat{\sigma}_i$ is in $[\sigma_i]$ and $\hat{a}_i = a_i$ and $\hat{\sigma}_i'$ is in $[\sigma_i']$, it is the case that $\hat{\rho}$ is a model of ψ.

We can now construct a non-deterministic Rabin tree automaton $\mathcal{R}_{ctrl}$ which will run over $\{\top, \bot\}$-labelled Δ_{RA}-trees (whose underlying tree is $\mathcal{T}$) $\mathcal{R}_{ctrl}$ accepts $(\mathcal{T}, \xi)$ iff $(\mathcal{T}, \xi)$ is an admission strategy tree representing a region-respecting ψ-winning admission strategy. It will be convenient to view $\mathcal{R}_{ctrl}$ as the intersection of three non-deterministic tree automata $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{R}_3$, where $\mathcal{B}_1$, $\mathcal{B}_2$ are non-deterministic Büchi tree automata and $\mathcal{R}_3$ is a deterministic Rabin tree automaton. For an input $\{\top, \bot\}$-labelled Δ_{RA}-tree $(\mathcal{T}, \xi)$, $\mathcal{B}_1$ checks whether $(\mathcal{T}, \xi)$ is an admission strategy tree, $\mathcal{B}_2$ examines whether the region-respecting admission strategy represented by $(\mathcal{T}, \xi)$ is safe, and $\mathcal{R}_3$ verifies that for every path ρ of $\mathcal{T}$ along which every node is labelled $\top$ by ξ, ρ is a model of ψ (in the sense defined above).

We have $\mathcal{B}_1 = (RG \times \{\top, \bot\}, ([\langle q_{in}, V_{in}, \varepsilon_{qu}, V_{in}^{qu}, \sharp\rangle], \top), \{\top, \bot\}, \hookrightarrow_1, RG \times \{\top, \bot\})$ where $\hookrightarrow_1$ is given by:

- Suppose $[\sigma]$ is in RG where σ is an environment configuration. Then $([\sigma], \top) \overset{\top}{\hookrightarrow}_1 \chi$, where $\chi : \mathit{Tran}([\sigma]) \to (RG \times \{\top, \bot\})$ is given by: $\chi(\langle[\sigma], a, [\sigma']\rangle) = ([\sigma'], \top)$.

  Further, $([\sigma], \bot) \overset{\bot}{\hookrightarrow}_1 \chi$, where $\chi : \mathit{Tran}([\sigma]) \to (RG \times \{\top, \bot\})$ is given by: $\chi(\langle[\sigma], a, [\sigma']\rangle) = ([\sigma'], \bot)$.

- Suppose $[\sigma]$ is in RG where σ is a system configuration. It follows that $\mathit{Tran}([\sigma]) = \{\langle[\sigma], m_1, [\sigma'_1]\rangle, \langle[\sigma], m_2, [\sigma'_2]\rangle\}$ where $m_1 = 0$ and $m_2 = 1$. We have $([\sigma], \top) \overset{\top}{\hookrightarrow}_1 \chi$ for any $\chi : \mathit{Tran}([\sigma]) \to (RG \times \{\top, \bot\})$ which maps each $\langle[\sigma], m_i, [\sigma'_i]\rangle$ to $([\sigma'_i], b_i)$, where $b_i \in \{\top, \bot\}$, and satisfies the condition that there exists at least one i with $b_i = \top$.

  Further, $([\sigma], \bot) \overset{\bot}{\hookrightarrow}_1 \chi$, where $\chi : \mathit{Tran}([\sigma]) \to (RG \times \{\top, \bot\})$ is given by: $\chi(\langle[\sigma], a, [\sigma']\rangle) = ([\sigma'], \bot)$.

- If $[\mathsf{Err}]$ is in RG, then we have $([\mathsf{Err}], \top) \overset{\top}{\hookrightarrow}_1 \chi$, where χ is the function from $\mathit{Tran}([\mathsf{Err}])$ to $RG \times \{\top, \bot\}$ that maps $\langle[\mathsf{Err}], \mathsf{err}, [\mathsf{Err}]\rangle$ to $([\mathsf{Err}], \bot)$.

Intuitively, for an input $\{\top, \bot\}$-labelled Δ_{RA}-tree $(\mathcal{T}, \xi)$, a state $([\sigma], b)$ of $\mathcal{B}_1$ indicates that $\mathcal{B}_1$ expects a node ρ in $\mathcal{T}$ such that $\mathit{dir}(\rho) = \langle[\hat{\sigma}], a, [\hat{\sigma}']\rangle$ with $[\hat{\sigma}'] = [\sigma]$, and $\xi(v) = b$. It is straightforward to verify that $\mathcal{B}_1$ accepts $(\mathcal{T}, \xi)$ iff $(\mathcal{T}, \xi)$ is an admission strategy tree.

We next define $\mathcal{B}_2 = \big(RG \times (\{\top, \bot\} \cup \{\$\}), ([\langle q_{in}, V_{in}, \varepsilon_{qu}, V^{qu}_{in}, \sharp\rangle], \top), \{\top, \bot\}, \hookrightarrow_2, RG \times \{\top, \bot\}\big)$ where $\$$ is a special symbol and $\hookrightarrow_2$ is given by:

- Suppose $[\sigma]$ is in RG and $\sigma \neq \mathsf{Err}$. Then we have $([\sigma], \top) \overset{\top}{\hookrightarrow}_2 \chi$ for any map $\chi : \mathit{Tran}([\sigma]) \to (RG \times (\{\top, \bot\} \cup \{\$\}))$.

  Further, we have $(\sigma, \bot) \overset{\bot}{\hookrightarrow}_2 \chi$, where χ is the function from $\mathit{Tran}([\sigma])$ to $RG \times (\{\top, \bot\} \cup \{\$\})$ defined by: $\chi(\langle[\sigma], a, [\sigma']\rangle) = ([\sigma'], \bot)$.

- Suppose $[\mathsf{Err}]$ is in RG. Firstly, we have $([\mathsf{Err}, \top]) \overset{\top}{\hookrightarrow}_2 \chi$, where χ is the function from $\mathit{Tran}([\mathsf{Err}])$ to $RG \times (\{\top, \bot\} \cup \{\$\})$ defined by: $\chi(\langle[\mathsf{Err}], \mathsf{err}, [\mathsf{Err}]\rangle) = ([\mathsf{Err}], \$)$.

  Secondly, $([\mathsf{Err}, \bot]) \overset{\bot}{\hookrightarrow}_2 \chi$, where χ is the function $\mathit{Tran}([\mathsf{Err}])$ to $RG \times (\{\top, \bot\} \cup \{\$\})$ given by: $\chi(\langle[\mathsf{Err}], \mathsf{err}, [\mathsf{Err}]\rangle) = ([\mathsf{Err}], \bot)$.

Lastly, $([\mathsf{Err},\$]) \overset{\perp}{\hookrightarrow}_2 \chi$, where χ is the function from $\mathit{Tran}([\mathsf{Err}])$ to $RG \times (\{\top,\perp\} \cup \{\$\})$ defined by: $\chi(\langle[\mathsf{Err}],\mathsf{err},[\mathsf{Err}]\rangle) = ([\mathsf{Err}],\$)$.

Similarly to $\mathcal{B}_1$, for an input $\{\top,\perp\}$-labelled Δ_{RA}-tree $(\mathcal{T},\xi)$, a state $([\sigma],b)$ of $\mathcal{B}_2$ indicates that $\mathcal{B}_2$ expects a node ρ in $\mathcal{T}$ such that $\mathit{dir}(\rho) = \langle[\hat{\sigma}],a,[\hat{\sigma}']\rangle$ with $[\hat{\sigma}'] = [\sigma]$, and $\xi(v) = b$. It is again clear that $\mathcal{B}_2$ accepts an admission strategy tree $(\mathcal{T},\xi)$ iff the admission strategy represented by $(\mathcal{T},\xi)$ is safe.

To define $\mathcal{R}_3$, we first note that for the LTL formula ψ, one can effectively construct a non-deterministic Büchi automaton $\mathcal{B}_\psi$ over 2^{AP} ([78]) which has the following property: Let σ be in $(2^{AP})^\omega$. Then σ is accepted by $\mathcal{B}_\psi$ iff σ is a model of ψ. From $\mathcal{B}_\psi$, we can construct a *deterministic* Rabin automaton $\mathcal{R}_\psi$ ([68]) over 2^{AP} such that: for every σ in $(2^{AP})^\omega$, σ is accepted by $\mathcal{B}_\psi$ iff σ is accepted by $\mathcal{R}_\psi$. The Rabin tree automaton $\mathcal{R}_3$ that we next construct will simulate $\mathcal{R}_\psi$ along every path of which every node is labelled $\top$. The crucial point to note is that $\mathcal{R}_\psi$ is deterministic.

We also recall from section 4.4 the operator Prj which maps transitions in Δ_{TS} to transitions in $\Delta_{\mathcal{A}}$. We extend Prj in the obvious way to Δ_{RA} as follows:

- Suppose $\langle[\sigma],a,[\sigma']\rangle$ where σ is an environment configuration. Then $\mathrm{Prj}(\langle[\sigma],a,[\sigma']\rangle) = \mathrm{Prj}(\delta)$ where $\delta = \langle\hat{\sigma},(t,\hat{a}),\hat{\sigma}'\rangle$ is in Δ_{TS} with $\hat{\sigma} \in [\sigma]$ and $\hat{a} = a$ and $\hat{\sigma}' \in [\sigma']$.
- Suppose $\langle[\sigma],m,[\sigma']$ where σ is a system configuration. Then $\mathrm{Prj}(\langle[\sigma],m,[\sigma']\rangle) = \mathrm{Prj}(\delta)$ where $\delta = \langle\hat{\sigma},\hat{m},\hat{\sigma}'\rangle$ is in Δ_{TS} with $\hat{\sigma} \in [\sigma]$ and $\hat{m} = m$ and $\hat{\sigma}' \in [\sigma']$.

It is easy to see that $\mathrm{Prj}(\delta)$ can be effectively found for any δ in Δ_{RA}.

Suppose $\mathcal{R}_\psi = (S_\psi, s^\psi_{in}, 2^{AP}, \hookrightarrow_\psi, \mathcal{F}_\psi)$, where $\mathcal{F}_\psi = \{(E_1,F_1),(E_2,F_2), \ldots,(E_k,F_k)\}$. We now define

$$\mathcal{R}_3 = (RG \times (S_\psi \cup \{\$\}),\ ([\langle q_{in}, V_{in}, \varepsilon_{qu}, V^{qu}_{in}, \natural\rangle], s^\psi_{in}),\ \{\top,\perp\},\ \hookrightarrow_3,\ \widehat{\mathcal{F}})$$

with $\widehat{\mathcal{F}} = \{\big(RG \times E_i, RG \times (F_i \cup \{\$\})\big) \mid i = 1, 2, \ldots, k\}$, where \$ is a special symbol not in S_ψ. And $\hookrightarrow_3$ is defined as follows: Suppose $[\sigma]$ is in RG. Then we have

- For $s \in S_\psi$, we have $([\sigma], s) \overset{\top}{\hookrightarrow}_3 \chi$, where χ is the function from $\mathit{Tran}([\sigma])$ to $RG \times (S_\psi \cup \{\$\})$ such that for each $\delta = \langle [\sigma], a, [\sigma'] \rangle$, we have $\chi(\delta) = ([\sigma'], s')$, where $s \overset{\lambda(\hat{\delta})}{\hookrightarrow}_\psi s'$ with $\hat{\delta} = \mathrm{Prj}(\delta)$.

  Further, $([\sigma], s) \overset{\perp}{\hookrightarrow}_3 \chi$, where χ is the function from $\mathit{Tran}([\sigma])$ to $RG \times (S_\psi \cup \{\$\})$ such that for each $\delta = \langle [\sigma], a, [\sigma'] \rangle$, we have $\chi(\delta) = ([\sigma'], \$)$.
- $([\sigma], \$) \overset{\top}{\hookrightarrow}_3 \chi$ and $([\sigma], \$) \overset{\perp}{\hookrightarrow}_3 \chi$, where χ is the function from $\mathit{Tran}([\sigma])$ to $RG \times (S_\psi \cup \{\$\})$ such that for each $\delta = \langle [\sigma], a, [\sigma'] \rangle$, we have $\chi(\delta) = ([\sigma'], \$)$.

Intuitively, the tree automaton $\mathcal{R}_3$ simulates $\mathcal{R}_\psi$ along the paths in which every node is labelled ρ and assign to each node in such a path the corresponding *unique* state reached by $\mathcal{R}_\psi$ upon reading the atomic propositions of v. The states of the form $([\sigma], \$)$ in $\mathcal{R}_3$, where $[\sigma] \in RG$, are used to indicate that the node being read is labelled $\perp$ and hence is irrelevant.

Since $\mathcal{R}_{\mathit{ctrl}}$ is the intersection of $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{R}_3$, it is now routine to verify that the language of $\mathcal{R}_{\mathit{ctrl}}$ is nonempty iff there exists a region-respecting ψ-winning admission strategy for $\mathcal{A}$. This establishes theorem 4.2(i), owing to lemma 4.5.

We analyze the complexity of the above decision procedure for theorem 5.4(i). The non-deterministic Büchi automaton $\mathcal{B}_\psi$ will have $N_{\mathcal{B}_\psi} = 2^{O(|\psi|)}$ states, where $|\psi|$ is the size of ψ. Hence the deterministic Rabin automaton for $\mathcal{R}_\psi$ has $N_{\mathcal{R}_\psi} = 2^{O(N_{\mathcal{B}_\psi} \cdot \log N_{\mathcal{B}_\psi})}$ states and $K_{\mathcal{R}_\psi} = O(N_{\mathcal{B}_\psi})$ accepting pairs ([68]). It is easy to construct the non-deterministic Rabin tree automaton $\mathcal{R}_{\mathit{ctrl}}$ directly which will have $N_{\mathcal{R}_{\mathit{ctrl}}} = O(|RG| \cdot N_{\mathcal{R}_\psi})$ states and $K_{\mathcal{R}_{\mathit{ctrl}}} = O(N_{\mathcal{B}_\psi})$ accepting pairs. It is not difficult to see that

$|RG| = O\big(N_{\mathcal{A}} \cdot N_{\mathcal{A}_{qu}} \cdot 2^{O(k\log(kc))} \cdot 2^{d}\big)$ ([7]), where $N_{\mathcal{A}}$ is the number of locations of the plant, $N_{\mathcal{Q}}$ is the number of locations of the queue automaton $\mathcal{A}_{qu}$, k is the total number of clock variables ($k = |X| + |X_{qu}|$), c is the largest constant appearing across all the clock constraints (in $\mathcal{A}$ and $\mathcal{A}_{qu}$) and d is the total number of difference constraints (in $\mathcal{A}$ and $\mathcal{A}_{qu}$). The complexity for testing nonemptiness of $\mathcal{R}_{ctrl}$ ([23]) is $\big(N_{\mathcal{R}_{ctrl}} \cdot K_{\mathcal{R}_{ctrl}}\big)^{O(K_{\mathcal{R}_{ctrl}})}$, that is,

$$\big(O(N_{\mathcal{A}} \cdot N_{\mathcal{A}_{qu}} \cdot 2^{O(k\log(kc))} \cdot 2^{d} \cdot 2^{O(|\psi| \cdot 2^{O(|\psi|)})}\big)^{O(2^{|\psi|})} .$$

4.5.4. Working with Zones. We have settled theorem 4.2(i) using regions. Here we argue that one can also prove theorem 4.2(i) using zones.

Let $\mathcal{B} = (Q, q_{in}, X, \{I_q\}_{q\in Q}, \Sigma, \longrightarrow)$ be a timed automaton. A clock zone is a union of clock regions. A clock zone Z can be effectively represented as a clock constraint φ in the sense that for every clock valuation V, V is in Z iff V satisfies φ. A zone is a pair (q, Z), where $q \in Q$ and Z a clock zone. It is shown in [11] that for the timed automaton $\mathcal{B}$, that may contain difference constraints, we can quotient $TS_{\mathcal{B}}$ into a finite transition system $ZA_{\mathcal{B}}$, called the *zone automaton* of $\mathcal{B}$. We have $ZA_{\mathcal{B}} = (ZN_{\mathcal{B}}, (q_{in}, \wedge_{x\in X} x = 0), \Sigma, \rightarrowtail_{\mathcal{B}})$, where $ZN_{\mathcal{B}}$ is a subset of zones, and $\rightarrowtail_{\mathcal{B}} \subseteq ZN_{\mathcal{B}} \times \Sigma \times ZN_{\mathcal{B}}$ the transition relation. We refer to [11] for the detailed definition of $ZA_{\mathcal{B}}$. For our purpose, it suffices to note that, similar to Proposition 4.1, $ZA_{\mathcal{B}}$ has the following property:

Proposition 4.6. *Let $ZA_{\mathcal{B}}$ be as given above and assume the associated notations. If $(q, V) \overset{t,a}{\Longrightarrow}_{\mathcal{B}} (q', V')$, and $(q1, V1)$ is in the same zone as (q, V), then there exists $t' \in \mathbb{R}_{+}$ such that $(q1, V1) \overset{t',a}{\Longrightarrow}_{\mathcal{B}} (q1', V1')$ and $(q1', V1')$ being in the same zone as (q', V').*

Note that the timed automaton $\mathcal{A}_{qu}$ contains bounded clock subtraction operations. The construction of $ZA_{\mathcal{B}}$ in [11] can be easily adapted to handle bounded clock subtractions [24].

Now we turn to the task plant $\mathcal{A}$ and show that theorem 4.2(i) can also be handled using zones. Following the above discussion, we can define the zone version of the $\mathcal{A}$, by easily adapting the construction of zone automata for timed automata. We can then define zone-respecting admission strategies, in exactly the same way as region-respecting admission strategies. Namely, we say an admission strategy f is zone-respecting iff for every ρ, $\hat{\rho}$ in $\mathit{Runs}(\mathit{TS})$, if ρ, $\hat{\rho}$ are in $\mathit{Runs}(f)$ and ρ, $\hat{\rho}$ "pass" through the same sequence of zones, then $f(\rho) = f(\hat{\rho})$.

As usual, fix the LTL formula ψ. It is easy to see that the cornerstone for the proof of lemma 4.5 is Proposition 4.1. In this respect, with Proposition 4.6, we can then show that, there exists a ψ-winning admission strategy for $\mathcal{A}$ iff there exists a zone-respecting ψ-winning admission strategy for $\mathcal{A}$. It now follows that we can then prove theorem 4.2(i) by working with the zone automaton of $\mathcal{A}$ and with zone-respecting strategies.

As for complexity, the zone automaton of the task plant $\mathcal{A}$ will usually be smaller than its region automaton. However, in the worst case, the zone automaton of $\mathcal{A}$ can be of the same size as the region automaton of $\mathcal{A}$. Hence, the worst case complexity of the decision procedure for theorem 4.2(i) based on the zone automataon of $\mathcal{A}$ would be the same as that based on the region automaton of $\mathcal{A}$.

4.5.5. Synthesis of Admission Controllers.

Here we prove theorem 4.2(ii). Recall the Rabin tree automaton $\mathcal{R}_{\mathit{ctrl}}$ constructed in the proof of theorem 4.2(i). Suppose the language of $\mathcal{R}_{\mathit{ctrl}}$ is nonempty. Then by Rabin's tree theorem [63], the decision procedure for testing the nonemptiness of the language of $\mathcal{R}_{\mathit{ctrl}}$ yields a *regular* $\{\top, \bot\}$-labelled tree $(\mathcal{T}, \xi)$ that is accepted by $\mathcal{R}_{\mathit{ctrl}}$. More precisely, the set of nodes in $\mathcal{T}$ that are labelled $\top$ by ξ is a regular subset of $\Delta_{RA}^{\star}$, and $(\mathcal{T}, \xi)$ is presented in the form of a finite transition system, which we shall extend to a timed automaton $\mathcal{C}$. The finite timed automaton $\mathcal{C}$ will form the ψ-controller we seek for $\mathcal{A}$.

Each location of $\mathcal{C}$ will be a node of $\mathcal{T}$ and it will be accompanied by a labelling function γ. The initial location will be ε with $\gamma(\varepsilon) = [\langle q_{in}, V_{in}, \varepsilon_{qu}, V_{in}^{qu}, \sharp\rangle]$, whereas to all other locations γ will assign a transition of RA of the form $\langle[\sigma], a, [\sigma']\rangle$. From the region $[\sigma']$, we can recover a clock constraint θ over the clocks $X \cup X_{qu}$ and view it as the clock invariant associated with the location. To be precise, we must introduce an extra clock $\hat{x}$ to ensure that the system states are urgent.

We convert each edge (l, l') of $\mathcal{C}$ to a transition as follows. Let $\gamma(l') = \langle[\sigma], b, [\sigma']\rangle$. If σ is an environment configuration and $\mathrm{Prj}(\gamma(l')) = (q, \varphi, a, Y, q')$ (in $\Delta_{\mathcal{A}}$), then we convert (l, l') to $(l, \mathsf{true}, a, Y \cup \{\hat{x}\}, l')$. If σ is a system configuration, then we convert (l, l') to $(l, \hat{x} = 0, b, \emptyset, l')$.

4.5.6. QPLTL Specifications.

Our goal here is to show that theorem 4.2 can be generalized to QoS specifications in QPLTL.

Theorem 4.7.

(i) *Given a task plant $\mathcal{A}$ and a QPLTL formula ψ. we can effectively determine whether there exists a ψ-controller for $\mathcal{A}$.*
(ii) *Moreover, whenever such a ψ-controller exists, one can effectively construct a ψ-controller in the form of a finite timed automaton.*

Proof. The only point to note is that we can effectively construct a nondeterministic Büchi automaton $\mathcal{B}_\psi$ over 2^{AP} which accepts precisely the set of models of ψ. The rest follows from the proof of theorem 4.2. □

As pointed out in section 4.1, QPLTL is strictly more expressive than LTL and hence theorem 4.7 allows us to handle more general QoS specifications.

4.6. Discussion

In the present setting we have assumed that the admission controller has read-only access to the clocks of the plant (and the clocks of the queue automaton). This is a justifiable assumption since the plant model merely describes the expected task arrival patterns and this information can be assumed to be available to the admission controller as well. Nevertheless, it will be perhaps worth exploring settings where the admission controller is endowed with its own clocks with pre-specified granularities.

It will be equally interesting to study admission controllers that can interact with the ready queue so that tasks can be rejected *after* they have been admitted to the ready queue. One can also relax the requirement that every accepted task instance must be completed before its deadline, by allowing an accepted task instance to miss its deadline. However, we associate penalties with deadline misses and consider QoS specifications that take these penalties into account.

It will also be worthwhile considering settings with multi-processors, and settings where tasks have precedence constraints, and settings with shared resources.

CHAPTER 5

Controller Synthesis for Restricted Differential Hybrid Automata (RDAs)

In this chapter, we study controller synthesis in the hybrid setting. We propose a class of hybrid automata with discrete time semantics, which we called *restricted differential hybrid automata (RDA)*. Our aim is to study controller synthesis for RDAs.

We give an overview of RDAs and our results in section 5.1. Subsequently we review related work in section 5.2. We define RDAs formally in section 5.3. In section 5.4, we prove that the language of control state sequences generated by an RDA is regular. This regularity result provides the foundation for studying controller synthesis with RDAs, though it is also of independent interest for verification of RDAs. We formulate, in section 5.5, the controller synthesis problem for RDAs. In section 5.6, we prove that if there is no sensing delay, then the controller synthesis problem for RDAs with LTL (linear time temporal logic), or more generally, QPLTL (quantified propositional LTL) specifications is decidable. Further, if there exists a controller, then we can effectively synthesize one in the form of a (finite) RDA with no sensing delay. We conclude with a discussion and prospects of future directions in section 5.7.

5.1. Overview

A hybrid automaton models a digital control system interacting with a continuous environment. The digital system measures, that is, observes, the values of the continuous variables via sensors and updates the rates of evolution of the continuous via actuators. Basically, a hybrid automaton is a finite state transition system augmented with a set of real-valued variables. Loosely speaking, the transition system represents the digital control system. To emphasize this, states of the transition system are called *control states* or *modes*. The variables model the continuous environment. At each control state, the variables evolve according to some differential equations. Each transition of the hybrid automaton is associated with a guard in the form of certain constraints on the variables. A transition can be taken only when its guard evaluates to true. To each control state, we may also associate a constraint, called the invariant. We demand that the hybrid automaton can stay at a control state only when its invariant is true.

The most basic question about hybrid automata is the *reachability* problem: Given a hybrid automaton and a designated control state q_f, determine whether there exists a finite run such that the last control state is q_f.

In the continuous time semantics, a transition may be taken at any real-valued time provided its guard is true. This equips hybrid automata with very expressive power. Consequently, the reachability problem is undecidable even for simple subclasses of hybrid automata where each continuous variable evolves at constant rates [32] ($dx/dt = c$). In [6, 32, 42, 43], the reachability problem is shown to be decidable for variants of hybrid automata that have the additional *resetting* feature. The resetting feature demands that during mode changes, the value of a variable must be reset nondeterministically to some real number from a given set. Of course, in many practical control systems, one can not directly reset the value of a variable. Rather we can only update the evolution rate of a variable and thus indirectly influence the value of a variable. The resetting feature can be interpreted as giving over-approximations of the possible values of variables at each control state. We believe that this feature severely limits the ability of modelling practical hybrid systems faithfully.

We mention that [44] studies hybrid automata which have no discrete dynamics (that is, there is only one single discrete control state) and whose continuous dynamics are governed by a linear differential equation. The emphasis of [44] is to, given a pre-specified possible initial values of the continuous variables, effectively compute and represent the sets of possible values of these continuous variables due to evolution according to *the* linear differential equation.

Under the continuous time semantics, the work [35] shows that the reachability problem is decidable for a subclass of hybrid automata where each continuous variable evolves at constant rates. And guards in a loop can *not* refer to variables with rates different from 1 (that is, $\dot{x} = c$, $c \neq 1$).

In view of the expressiveness of the continuous time semantics, [30] proposes to study hybrid automata with discrete time semantics. The discrete time semantics demands that transitions can only be taken at integer-valued time instants. As argued in [27], if the hybrid automaton models the closed loop system consisting of a digital controller interacting with a continuous plant, then the discrete time semantics is the natural one; the digital controller will observe via sensors, the states of the plant and effect, via actuators, changes in the plant dynamics at discrete time instants determined by its internal clock. In [30], it is shown the reachability problem is decidable for the class of hybrid automata where the rate of each variable could be any constant from given bounded intervals. And values of variables are required to stay within a prescribed range. Further, each transition guard and invariant is a conjunction of rectangular constraints, where a rectangular constraint is a simple linear inequality involving just one variable.

The work [3] also considers the discrete time semantics and generalize it to allow sensing of values of variables and updating of rates of variables to both occur within bounded delays from integer time instants. For convenience, we shall call this the generalized discrete time semantics. The motivation comes from the fact that in practical hybrid systems, sensors and actuators do incur delays. For, it is reasonable to assume that there might be a delay between the time a sensor measured the value of a variable and the time the control system received this measurement. Further, when an actuator initiates an update on rates of variables, the rate change would generally not take effect immediately, but with some delay.

The main result of [3] is that the control state sequence languages of a class of hybrid automata are regular, under the generalized discrete time semantics. The features of this class of hybrid automata are as follows. Firstly, variables evolve at constant rates. Secondly, values of variables must stay within a prescribed range. Lastly, guards and invariants are rectangular.

The work [4] carries on the line of [3] and proves that, if one assumes the additional *finite precision* feature, then the regularity result in [3] also holds for the case that guards and invariants are *polynomial constraints*. The finite precision feature demands that values of variables can only be measured with a bounded accuracy. More precisely, there is a fixed rational ϵ such that, any value in the half-open interval $[(\ell - \frac{1}{2})\epsilon,\ (\ell + \frac{1}{2})\epsilon)$ is measured as $\ell\epsilon$. A *polynomial constraint* is a conjunction of polynomial inequalities in several variables.

In this thesis, we carry along the line of [3] and propose the class of hybrid automata which we called, *restricted differential hybrid automata (RDA)*. As in [3], we consider the generalized discrete time semantics. The key features of RDAs are as follows. Firstly, the rates of variables are governed by linear differential operators in *diagonal* forms. In other words, each variable may evolve at constant rates, or at exponential rates ($dx/dt = c \cdot x$). However, we further require that each variable either evolves at (possibly different) constant rates in all the control states or at (possibly different) exponential rates in all the control states. Secondly, the absolute values of variables must lie in a prescribed range. Lastly, guards and invariants are rectangular constraints. Note that we do not introduce the finite precision feature.

We prove that the language of control state sequences generated by an RDA is regular and can be effectively computed. We then use this regularity result to study controller synthesis with RDAs. We note that this regularity result is also of independent interest for verification of RDAs.

The work [2] studies the variant of RDAs with the finite precision feature and that guards and invariants are allowed to be polynomial guards. It is shown in [2] that the control state sequences languages of this variant of RDAs are regular. However, [2] does not investigate the controller synthesis problem.

We view an RDA naturally as a *plant*, which describes the possible interactions between the continuous environment and a digital control system. At each discrete time instant, the system receives measurements of the values of the variables and may decide to remain at the current control state (thus maintains the current rates of continuous variables) or to switch to another control state (thus updates the rates of continuous variables). A *strategy* should make recommendations for the system on which actions to take, based on the history of control states and measurements of variables in the past integer time instants. A strategy f is non-blocking iff the system is always able to make a move if it always follow the recommendations of f.

We are interested in controller synthesis of RDAs with general specifications. We label the control states of the RDA with atomic propositions. And consider linear time specifications given as LTL formulae, or more generally, QPLTL formulae. Such a specification dictates the desired set of infinite control state sequences.

We say a strategy is winning iff it is non-blocking and the infinite control state sequences generated by the controlled RDA satisfies the the given formula in LTL or QPLTL. By a controller, we mean a winning strategy.

We prove that if there is no sensing delay, then the controller synthesis for RDAs is decidable for LTL specifications, and more generally QPLTL specifications. Further, if there exists a controller, then we can effectively synthesize one in the form of a (finite) RDA with no sensing delay.

We do not know how to solve the controller synthesis problem for RDAs when sensing delays are present. The key obstacle is: At each integer time instant, the measurements of variables received by the system and the strategy are *not* the current values of variables. On the other hand, if there is no sensing delay, then a strategy knows precisely the current values of variables.

5.2. Related Work

Much of the literature in controller synthesis deal with only *safety* specifications. A safety specification typically demands that the hybrid plant should be controlled in such a way that it stays only in good control states (thus avoiding bad states) We refer to [8, 36, 70, 75, 76] for a representative sample of hybrid controller synthesis for safety specifications. The emphasis of this line of work is to be able to handle hybrid automata of rich continuous dynamics but derives *approximation* algorithms for computing the hybrid controller. In contrast, in this thesis, we are interested decidability results in controller synthesis for hybrid automata with respect to general specifications given by LTL or QPLTL formulae.

Controller synthesis for the class of rectangular hybrid automata against LTL specifications were studied in [29, 31]. In a rectangular hybrid automaton, the guards and invariants are rectangular constraints. And the rates of variables are constants that may vary in a given range.

In particular, [31] studies the discrete time setting and [29] investigates the continuous time setting. In both cases, the corresponding controller synthesis problem is shown to be decidable. Further, if a controller exists, then we can effectively construct one in the form of a rectangular hybrid automaton.

The work [57] studies the following controller synthesis for hybrid automata: Let $\mathcal{A}$ be a hybrid automaton whose continuous state space is partitioned into finitely many portions and each such portion is called an event. Given a specification in terms of a set of finite sequences of events, find a controller such that the trajectories of the controlled hybrid automaton traverse the given event sequences. Approximate approaches for solving this controller synthesis problem are then developed in [57]. We mention that though [57] uses modal logics ([21]). It use them merely for the development of these approximate solutions, and not as the formalism for the specification of desired behaviour.

5.3. Restricted Differential Hybrid Automata

Through the rest of this chapter, we fix a positive integer n and one function symbol x_i for each i in $\{1, 2, \ldots, n\}$. We will often refer to the x_i's as variables and will view each x_i as a function of *time* $x_i : \mathbb{R}_{\geq 0} \to \mathbb{R}$ with $\mathbb{R}$ being the set of reals and $\mathbb{R}_{\geq 0}$, the set of non-negative reals. We write $\dot{x}_i$ for the derivative of x_i over time. We let $\mathbb{Q}$ denote the set of rationals.

A *rectangular constraint* is $x_i \prec c$, where $i \in \{1, 2, \ldots, n\}$, $\prec$ is in $\{<, \leq, >, \geq\}$ and $c \in \mathbb{Q}$. A *rectangular guard* is a finite conjunction of rectangular constraints. We let *Grd* denote the set of rectangular guards. We will often refer to rectangular constraints (respectively, rectangular guards) simply as constraints (respectively, guards).

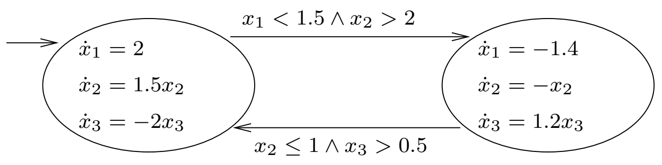
A *valuation* V is just a member of $\mathbb{R}^n$. It will be viewed as prescribing the value $V(i)$ to each variable x_i. The notion of a valuation satisfying a guard is defined in the obvious way. Namely, the valuation V satisfies the constraint $x_i \prec c$ iff $V(i) \prec c$. And V satisfies the guard g iff V satisfies every constraint in g.

A *restricted differential hybrid automaton (RDA)* is a structure $\mathcal{A} = (Q, q_{in}, V_{in}, \mathit{Delay}, \{\rho_q\}_{q\in Q}, \{\mathit{CON}, \mathit{DIF}\}, \{I_q\}_{q\in Q}, \{\gamma_{min}, \gamma_{max}\}, \longrightarrow)$ where:

- Q is a finite set of *control states* with q, q' ranging over Q.
- $q_{in} \in Q$ is the initial control state.
- $V_{in} \in \mathbb{Q}^n$ is the initial valuation.
- $\mathit{Delay} = (\delta^0_{ob}, \delta^1_{ob}, \delta^0_{up}, \delta^1_{up})$ where $\delta^0_{ob}, \delta^1_{ob}, \delta^0_{up}, \delta^1_{up} \in \mathbb{Q}$ are *delay parameters* such that $0 \leq \delta^0_{up} \leq \delta^1_{up} < \delta^0_{ob} \leq \delta^1_{ob} \leq 1$.
- $\{\rho_q\}_{q\in Q}$ is a family of rate functions associated with the control states. For each control state q, ρ_q is of the form $\dot{x} = A_q x + b_q$ where $x = (x_1, x_2, \ldots, x_n)$, A_q is a *diagonal* $n \times n$ matrix with rational entries and $b_q \in \mathbb{Q}^n$. Thus at control state q, each variable x_i evolves at the rate $\rho_q(i)$ given by: $\dot{x}_i = A_q(i,i) \cdot x_i + b_q(i)$.
- $\{\mathit{CON}, \mathit{DIF}\}$ is a partition of the indices $\{1, 2, \ldots, n\}$. And for every control state q, it is the case that: $i \in \mathit{CON}$ implies $A_q(i,i) = 0$; $i \in \mathit{DIF}$ implies $b_q(i) = 0$.

  Hence at each control state q, if $i \in \mathit{CON}$, then $\rho_q(i)$ is $\dot{x}_i = b_q(i)$. if $i \in \mathit{DIF}$, then $\rho_q(i)$ is $\dot{x}_i = A_q(i,i) \cdot x_i$. Intuitively, variables in $\{x_i \mid i \in \mathit{CON}\}$ must always evolve at constant rates, while variables in $\{x_i \mid i \in \mathit{DIF}\}$ must always evolve at differential (more precisely, exponential) rates.
- $\{I_q\}_{q\in Q}$ is a family of *invariants* associated with the control states. For each control state q, $I_q \in \mathit{Grd}$.
- $\gamma_{min}, \gamma_{max} \in \mathbb{Q}$ are *range parameters* such that $0 < \gamma_{min} < \gamma_{max}$.
- $\longrightarrow \subseteq Q \times \mathit{Grd} \times Q$ is a transition relation such that $q \neq q'$ for every (q, g, q') in $\longrightarrow$.

Through the rest of this chapter we fix an RDA $\mathcal{A}$ and assume its associated notations and terminology as defined above. An example of an RDA is given in figure 5.1, where all the control state invariants are true.



$\delta^0_{ob} = 0.8,\ \delta^1_{ob} = 0.9,\ \delta^0_{up} = 0.1,\ \delta^1_{up} = 0.2$
$V_{in}(1) = V_{in}(2) = V_{in}(3) = 1$
$\gamma_{min} = 0.5,\ \gamma_{max} = 4$

FIGURE 5.1. An RDA

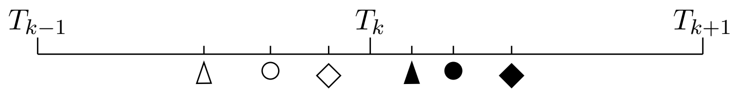
In what follows, we explain the intuition of the parameters of $\mathcal{A}$ and give the operational semantics of $\mathcal{A}$. As in [3], we assume the discrete time semantics, but allow delays in both sensing of values of variables and updating of rates of variables. For clarity, we will write T_k for the time instant k, for $k = 0, 1, 2, \ldots$.

Suppose at time instant T_{k-1}, the RDA $\mathcal{A}$ was at control state q. Then at time T_k, $\mathcal{A}$ may remain at q if the n-tuple $(v_1, v_2, \ldots, v_n)$ of the values of x_i's satisfies I_q. Intuitively, I_q dictates the condition on the current values of x_i's, in order for $\mathcal{A}$ to stay at q.

At each time instant T_k $(= k)$, the RDA $\mathcal{A}$ receives a measurement regarding the current values of the x_i's. However, the value of x_i that is observed at time T_k is the value that held at some time $T_{k-1} + t^{ob}_i$ where $t^{ob}_i \in [\delta^0_{ob},\ \delta^1_{ob}]$. If at time instant T_k, $\mathcal{A}$ is in control state q and (q, g, q') is a transition in $\mathcal{A}$, then $\mathcal{A}$ may perform this transition instantaneously and move to the control state q', provided the following conditions hold:

- The *observed* n-tuple of values $(v_1, v_2, \ldots, v_n)$ received at T_k satisfies the guard g.
- The current n-tuple of (*actual*) values of variables satisfies the invariant $I_{q'}$.

If $\mathcal{A}$ takes the transition (q, g, q'), then as a result, each x_i will cease to evolve according to the rate $\rho_q(i)$ and instead start evolving according to the rate $\rho_{q'}(i)$. However, this change in the rate of evolution of each x_i will not kick in at T_k but at some time $T_k + t_i^{up}$ where $t_i^{up} \in [\delta_{up}^0,\ \delta_{up}^1]$. We illustrate the relative positions of t_i^{ob}, t_i^{up}, for a fixed i, and the delay bounds in figure 5.2. We emphasize that the reals t_i^{ob}, $i = 1, 2, \ldots, n$, are not necessarily identical. Also the reals t_i^{up}, $i = 1, 2, \ldots, n$, are not necessarily identical.



△ : $T_{k-1} + \delta_{ob}^0$
○ : $T_{k-1} + t_i^{ob}$
◇ : $T_{k-1} + \delta_{ob}^1$
▲ : $T_k + \delta_{up}^0$
● : $T_k + t_i^{up}$
◆ : $T_k + \delta_{up}^1$

FIGURE 5.2. Delay parameters

Intuitively, both the sensing of values of the x_i's and the rate changes associated with mode switching take place in a lazy fashion but with bounded delays. We expect $\delta_{ob}^0, \delta_{ob}^1$ to be close to 1 and $\delta_{up}^0, \delta_{up}^1$ to be close to 0 while both $\delta_{ob}^1 - \delta_{ob}^0$ and $\delta_{up}^1 - \delta_{up}^0$ to be small compared to 1. In the idealized setting, the value observed at T_k is the value that holds at exactly T_k ($\delta_{ob}^0 = 1 = \delta_{ob}^1$) and the change in rates due to mode switching would kick in immediately ($\delta_{up}^0 = 0 = \delta_{up}^1$).

The parameters $\gamma_{min}, \gamma_{max}$ specify the relevant range of the absolute values of the variables. The RDA $\mathcal{A}$ gets stuck if for some i, $|x_i|$ gets outside the allowed range $[\gamma_{min}, \gamma_{max}]$. Loosely speaking, the γ_{max} bound is used to restrict the amount of information carried by a variable evolving at a (positive or negative) constant rate, and a variable increasing at an exponential rate ($\dot{x}_i = c \cdot x_i$, $c > 0$). On the other hand, γ_{min} is used to restrict the amount of information carried by a variable decreasing at an exponential rate ($\dot{x}_i = c \cdot x_i$, $c < 0$). We note that our setting is quite different from the classical continuous setting. Hence the standard control objective of driving a system variable to 0 is not relevant here and thus does not pose a serious limitation.

The behaviour of $\mathcal{A}$ will be defined in terms of an associated transition system. We assume that the unit of time has been fixed at some suitable level of granularity and that the rate functions $\{\rho_q\}_{q \in Q}$ have been scaled accordingly.

A *configuration* is a triple (q, V, q') where q, q' are control states and V is a valuation. q is the current control state, q' is the control state that held at the previous time instant, and V captures the *actual* values of the variables at the current time instant. The valuation V is said to be *feasible* if $\gamma_{min} \leq |V(i)| \leq \gamma_{max}$ for every i in $\{1, 2, \ldots, n\}$. The configuration (q, V, q') is *feasible* iff V is a feasible valuation.

The initial configuration is by convention (q_{in}, V_{in}, q_{in}) and is assumed to be feasible. Further, we assume V_{in} satisfies $I_{q_{in}}$. We let $Conf_{\mathcal{A}}$ denote the set of configurations.

Suppose the RDA $\mathcal{A}$ was in the configuration $(q_{k-1}, V_{k-1}, q'_{k-1})$ at time T_{k-1} and one unit of time has elapsed. Then at the current time instant T_k, $\mathcal{A}$ will make an instantaneous move by executing a transition or the silent action τ and move to a configuration (q_k, V_k, q'_k). The silent action τ will be used to record that no mode change has taken place during this move. The action μ will be used to record that a transition has been taken and as a result, a mode change has taken place. As is common, we will collapse the unit-time-passage followed by an instantaneous transition into one "time-abstract" transition labelled by τ or μ.

We wish to formalize the transition relation $\Longrightarrow$, which is a subset of $Conf_{\mathcal{A}} \times \{\tau, \mu\} \times Conf_{\mathcal{A}}$. For doing so, we note a basic fact. Let q be a control state and i be in $\{1, 2, \ldots, n\}$. Given a real number v and a real T, we have:

- If $i \in CON$, then the curve $x_i : [0,T] \to \mathbb{R}$ with $x_i(t) = v + b_q(i) \cdot t$ is the unique curve which satisfies $x_i(0) = v$ and $dx_i/dt = b_q(i)$.
- If $i \in DIF$, then the curve $x_i : [0,T] \to \mathbb{R}$ with $x_i(t) = v \cdot \exp(A_q(i,i) \cdot t)$ is the unique curve (see [33]) which satisfies $x_i(0) = v$ and $dx_i/dt = A_q(i,i) \cdot x_i$.

In what follows, we shall denote the value $x_i(T)$ thus obtained by $Val(q,i,T,v)$.

The transition relation $\Longrightarrow$ is the least subset of $Conf_{\mathcal{A}} \times \{\tau, \mu\} \times Conf_{\mathcal{A}}$ which satisfies the following condition. Let (q,V,q'), $(q1,V1,q1')$ be in $Conf_{\mathcal{A}}$. Suppose there exist reals t_i^{up} in $[\delta_{up}^0, \delta_{up}^1]$, t_i^{ob} in $[\delta_{ob}^0, \delta_{ob}^1]$, $i = 1,2,\ldots,n$, such that for $i = 1,2,\ldots,n$, we have $V1(i) = Val(q,\, i,\, 1-t_i^{up},\, w_i)$ where $w_i = Val(q',\, i,\, t_i^{up},\, V(i))$.

- If $q1 = q1' = q$ and $V1(i)$ satisfies the invariant I_q. Then $(q,V,q') \xRightarrow{\tau} (q1,V1,q1')$.
- Suppose $q1' = q$ and there exists a transition $(q,g,q1)$ in $\longrightarrow$ such that:
  - The valuation U satisfies the guard g, where $U(i) = Val(q,\, i,\, t_i^{ob} - t_i^{up},\, w_i)$, for $i = 1,2,\ldots,n$.
  - $V1(i)$ satisfies the invariant I_{q1}.

  Then $(q,V,q') \xRightarrow{\mu}_{\mathcal{A}} (q1,V1,q1')$.

Basically we have the four possible transition types as follows:

Type (i): $q = q'$ and $\alpha = \tau$.

Since $q = q'$, no mode change has taken place at the previous time instant T_{k-1}. Hence, during the interval $[T_{k-1},\, T_k)$, each variable x_i evolves according to $\rho_{q'}(i)$. Since $\alpha = \tau$, there will also be no mode change at the current time instant T_k. It follows during the interval $[T_k, T_{k+1})$, each variable x_i evolves at the rate $\rho_q(i)$, that is, $\rho_{q'}(i)$.

Type (ii): $q = q'$ and $\alpha = \mu$.

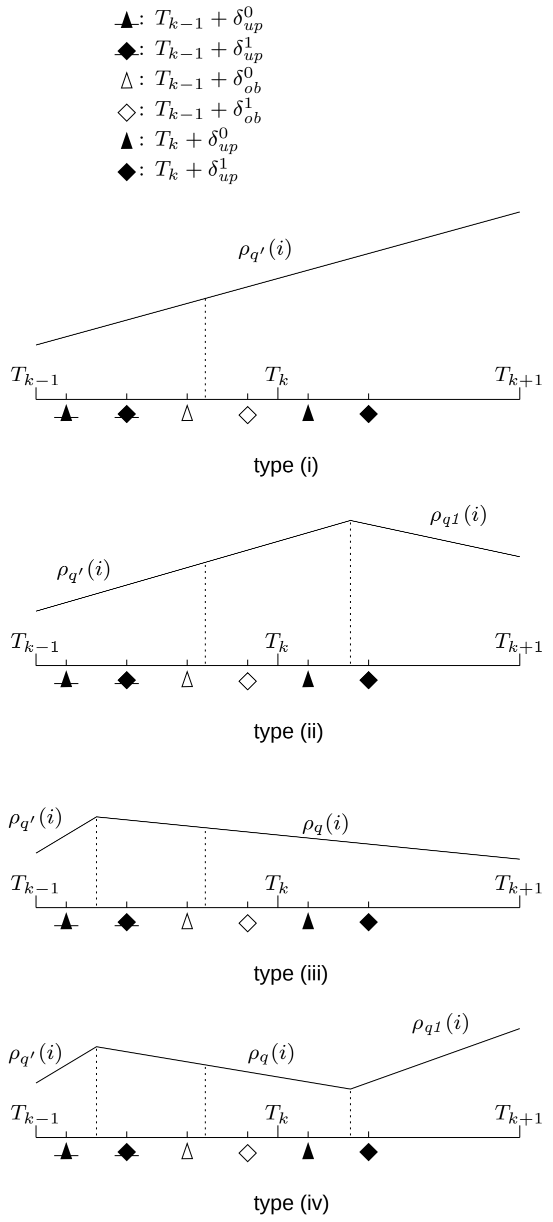


FIGURE 5.3. Four transition types

As with type (i), no mode change took place at the previous time instant T_{k-1} and thus each variable x_i evolve according to $\rho_{q'}(i)$ during the interval $[T_{k-1},\ T_k)$. However $\mathcal{A}$ makes a mode change at the current time instant T_k. Hence for each variable x_i, there exists a real $s_i^{up} \in [\delta^0_{up},\ \delta^1_{up}]$ such that x_i evolves at the rate $\rho_q(i)$ during the interval $[T_k,\ T_k + s_i^{up})$, but it assumes the rate $\rho_{q1}(i)$ during the interval $[T_k + s_i^{up},\ T_{k+1})$.

Type (iii): $q \neq q'$ and $\alpha = \tau$.

Since $q \neq q'$, a mode change has taken place at the previous time instant T_{k-1}. Hence for each variable x_i, there exists a real $t_i^{up} \in [\delta_{up}^0,\ \delta_{up}^1]$ such that x_i evolves at the rate $\rho_{q'}(i)$ during the interval $[T_{k-1},\ T_{k-1} + t_i^{up})$, but it assumes the rate $\rho_q(i)$ during the interval $[T_{k-1} + t_i^{up},\ T_k)$.

Since $\alpha = \tau$, there is no mode change occurs at the current time instant T_k. It follows that each x_i continues to evolve at the rate $\rho_q(i)$ during the interval $[T_k,\ T_{k+1})$.

Type (iv): $q \neq q'$ and $\alpha = \mu$.

As with type (iii), a mode change has occurred at the previous time instant T_{k-1}. Hence each x_i evolves according to $\rho_{q'}(i)$ during the interval $[T_{k-1},\ T_{k-1} + t_i^{up})$, but at the rate $\rho_q(i)$ during the interval $[T_{k-1} + t_i^{up},\ T_k)$, where t_i^{up} is a real in $[\delta_{up}^0,\ \delta_{up}^1]$.

Since $\alpha = \mu$, a mode change also takes place at the current time instant T_k. Hence, for each variable x_i, there exists $s_i^{up} \in [\delta_{up}^0,\ \delta_{up}^1]$ such that x_i evolves at the rate $\rho_q(i)$ during the interval $[T_k,\ T_k + s_i^{up})$, but at the rate $\rho_{q1}(i)$ during the interval $[T_k + s_i^{up},\ T_{k+1})$.

Figure 5.3 illustrates the four transition types by depicting the evolution of a variable x_i with $i \in CON$. For convenience, in the definition of $\Longrightarrow$, we have collapsed these four transition types into two cases according to τ or μ being the action label, and in each case have handled the subcases $q = q'$ and $q \neq q'$ simultaneously.

Now we define the transition system $TS_{\mathcal{A}} = (RC_{\mathcal{A}}, (q_{in}, V_{in}, q_{in}), \{\tau, \mu\}, \Longrightarrow_{\mathcal{A}})$ associated with $\mathcal{A}$ via:

- $RC_{\mathcal{A}}$, the set of *reachable configurations* of $\mathcal{A}$ is the least subset of $Conf_{\mathcal{A}}$ that contains the initial configuration (q_{in}, V_{in}, q_{in}) and satisfies: Suppose (q, V, q') is in $RC_{\mathcal{A}}$ and is a feasible configuration. Suppose further, $(q, V, q') \overset{\alpha}{\Longrightarrow} (q1, V1, q1')$ for some $\alpha \in \{\tau, \mu\}$. Then $(q1, V1, q1') \in RC_{\mathcal{A}}$.

- $\Longrightarrow_{\mathcal{A}}$ is $\Longrightarrow$ restricted to $RC_{\mathcal{A}} \times \{\tau, \mu\} \times RC_{\mathcal{A}}$.

We note that a reachable configuration can be the source of a transition in $TS_{\mathcal{A}}$ only if it is feasible. Thus infeasible reachable configurations will be deadlocked in $TS_{\mathcal{A}}$. A *run* of $TS_{\mathcal{A}}$ is a finite sequence of the form

$$\sigma = (q_0, V_0, q'_0)\,(q_1, V_1, q'_1)\;\ldots\;(q_\ell, V_\ell, q'_\ell)$$

where (q_0, V_0, q'_0) is the initial configuration and for each $k = 0, 1, \ldots, \ell - 1$, there exists $\alpha_k \in \{\tau, \mu\}$ with $(q_k, V_k, q'_k) \overset{\alpha_k}{\Longrightarrow}_{\mathcal{A}} (q_{k+1}, V_{k+1}, q'_{k+1})$. By a run of $\mathcal{A}$, we shall mean a run of $TS_{\mathcal{A}}$. We let $Runs(\mathcal{A})$ denote the set of runs of $\mathcal{A}$. The *control state sequence* induced by the run σ above is the sequence $q_0\, q_1\, \ldots\, q_\ell$. We define the *state sequence language* of $\mathcal{A}$, denoted $L_{st}(\mathcal{A})$, to be the set of control state sequences induced by runs of $\mathcal{A}$. We will often refer to the state sequence language of $\mathcal{A}$ simply as the *language* of $\mathcal{A}$.

5.4. State Sequence Languages of RDAs

Here we study the state sequence languages of RDAs. Our main result is:

Theorem 5.1. *Let $\mathcal{A}$ be an RDA. Then $L_{st}(\mathcal{A})$ is a regular subset of $Q^\star$. Further, a finite state automaton accepting $L_{st}(\mathcal{A})$ can be effectively computed from $\mathcal{A}$.*

Theorem 5.1 will provide the foundation for studying controller synthesis problems with RDAs. Independently, theorem 5.1 also implies various model checking problems for RDAs, in particular the reachability problem, can be effectively solved.

The rest of this section is devoted the proof of theorem 5.1. This proof consists of two major steps. The first one is to quotient the set of reachable configurations $RC_{\mathcal{A}}$ into a *finite number* of equivalence classes using a suitably chosen equivalence relation $\approx$. The crucial property required of $\approx$ is that it should be a congruence with respect to the transition relation of

$TS_{\mathcal{A}}$. In other words, if $(q1, V1, q1') \approx (q2, V2, q2')$ and $(q1, V1, q1') \overset{\alpha}{\Longrightarrow}_{\mathcal{A}}$ $(q3, V3, q3')$, then we require that there exists a configuration $(q4, V4, q4')$ such that $(q2, V2, q2') \overset{\alpha}{\Longrightarrow}_{\mathcal{A}} (q4, V4, q4')$ *and* $(q3, V3, q3') \approx (q4, V4, q4')$. The second step is to show that we can effectively compute these equivalence classes and a transition relation over them such that the resulting finite state automaton generates the language of $\mathcal{A}$.

For notational convenience, we assume $V_{in}(i) > 0$ for every $i \in DIF$. It is clear that this involves no loss of generality. The key consequence of this assumption is that in any reachable configuration of $\mathcal{A}$, the value of x_i for $i \in DIF$ will be positive.

We also assume without loss of generality that for each constraint $x_i \prec c$ that appeared in control state invariants and transition guards of $\mathcal{A}$, it is the case that $|\gamma_{min}| \le c \le |\gamma_{max}|$ if $i \in CON$ and $\gamma_{min} \le c \le \gamma_{max}$ if $i \in DIF$.

Define Δ to be the largest rational which *integrally* divides every member of $\{\delta^0_{ob}, \delta^1_{ob}, \delta^0_{up}, \delta^1_{up}, 1\}$. Let RAT be the set of rational numbers given by: $c \in RAT$ iff there exists a constraint $x_i \prec c$ which appears in some control state invariant or transition guard of $\mathcal{A}$. Define Γ to be the largest rational which *integrally* divides every number in the finite set of rational numbers

$$\begin{aligned} &\{A_q(i,i) \cdot \Delta \mid q \in Q, i \in DIF\} \\ \bigcup\ &\{b_q(j) \cdot \Delta \mid q \in Q, j \in CON\} \\ \bigcup\ &\{\gamma_{min}, \gamma_{max}\} \\ \bigcup\ &RAT\ . \end{aligned}$$

Define Θ_{con} to be the finite set of rational numbers

$$\begin{aligned} &\{h\Gamma \in [-\gamma_{max}, -\gamma_{min}] \mid h \in \mathbb{Z}\} \\ \bigcup\ &\{h\Gamma \in [\gamma_{min}, \gamma_{max}] \mid h \in \mathbb{Z}\} \end{aligned}$$

where $\mathbb{Z}$ is the set of integers. In other words, Θ_{con} contains integral multiples of Γ in the intervals $[-\gamma_{max}, -\gamma_{min}]$ and $[\gamma_{min}, \gamma_{max}]$.

Let RAT_{dif} be the finite set of rational numbers given by: $c \in RAT_{dif}$ iff there exists a constraint $x_i \prec c$ that appears in some control state invariant or transition guard of $\mathcal{A}$ *and* $i \in \mathit{DIF}$. Let Θ_{IR} be the set of irrational numbers $\{\ln \gamma_{min}, \ln \gamma_{max}\} \cup \{\ln c \mid c \in \mathit{RAT}_{\mathit{dif}}\}$. Define Θ_{dif} to be the *finite* set of real numbers

$$\{h\Gamma \in [\ln \gamma_{min}, \ln \gamma_{max}] \mid h \in \mathbb{Z}\}$$
$$\bigcup \{\ell\Gamma + \theta \in [\ln \gamma_{min}, \ln \gamma_{max}] \mid \ell \in \mathbb{Z},\, \theta \in \Theta_{\mathit{IR}}\} .$$

In other words, Θ_{dif} contains rational numbers of the form $h\Gamma$ in the interval $[\ln \gamma_{min}, \ln \gamma_{max}]$ where h is a positive integer, and irrational numbers of the form $\ell\Gamma + \theta$ in the interval $[\ln \gamma_{min}, \ln \gamma_{max}]$ where ℓ is an integer, that may be positive, zero or negative, and θ is a member of Θ_{IR}.

Loosely speaking, the set Θ_{con} (respectively Θ_{dif}) contains bounds relevant to the values of variables x_i's for $i \in \mathit{CON}$ (respectively $i \in \mathit{DIF}$). The points in Θ_{con} (Θ_{dif}) cut the real line into a finite number of segments. We shall use this segmentation to in turn partition the set of reachable configurations into finitely many equivalence classes. The simple but key observation that enables this is, in the (natural) logarithmic scale, exponential rates get represented as *constant* rates. More precisely, at control state q, if the rate of x_i is $\dot{x}_i = A_q(i,i) \cdot x_i$, then the derivative of $\ln x_i$ over time is the constant $A_q(i,i)$. However, if we were to deal with $\ln x_i$ directly instead of x_i, then we need to handle inequalities of the form $\ln x_i \prec \ln c$, that arises from constraints of the form $x_i \prec c$. Hence, the crux is to be able to handle *irrational* bounds $\ln c$.

In this light, let the members of Θ_{dif} be $\{\theta_1, \theta_2, \ldots, \theta_{|\Theta_{\mathit{dif}}|}\}$ where $\theta_1 < \theta_2 < \cdots < \theta_{|\Theta_{\mathit{dif}}|}$. We define the finite set of intervals $\mathcal{I}_{\mathit{dif}} = \{(-\infty, \theta_1), (\theta_1, \theta_2), \ldots, (\theta_{|\Theta_{\mathit{dif}}|-1}, \theta_{|\Theta_{\mathit{dif}}|}), (\theta_{|\Theta_{\mathit{dif}}|}, \infty)\} \bigcup \{[\theta_i, \theta_i] \mid i = 1, 2, \ldots, |\Theta_{\mathit{dif}}|\}$. In the same way, we define $\mathcal{I}_{con}$ from Θ_{con}.

Define the map $\|\cdot\|_{\mathit{dif}} : \mathbb{R}_+ \to \mathcal{I}_{\mathit{dif}}$ via: $\|v\| = I$ if $\ln v \in I$. Define $\|\cdot\|_{con} : \mathbb{R} \to \mathcal{I}_{con}$ via: $\|v\| = I$ if $v \in I$. Finally we define the map $\|\cdot\| : \mathit{RC} \to (\mathcal{I}_{\mathit{dif}} \cup \mathcal{I}_{con})^n$ by: $\|V\| = (I_1, I_2, \ldots, I_n)$ where $I_i = \|V(i)\|_{\mathit{dif}}$ for $i \in \mathit{DIF}$ and $I_i = \|V(i)\|_{con}$ for $i \in \mathit{CON}$.

We can now define the equivalence relation $\approx \subseteq RC \times RC$ by: $(q1, V1, q1') \approx (q2, V2, q2')$ iff $q1 = q2$, $\|V1\| = \|V2\|$ and $q1' = q2'$. The crucial property of $\approx$ is that $\approx$ is a congruence relation with respect to the transition relation $\Longrightarrow_{\mathcal{A}}$. In order to prove this, we first note the basic properties of $\|\cdot\|$.

Observation 5.2. *Let V, V' be valuations of $\mathcal{A}$ such that $\|V\| = \|V'\|$. Then we have:*

(i) *V is feasible iff V' is feasible.*

(ii) *Suppose V, V' are feasible. Let φ be a control state invariant or a transition guard of $\mathcal{A}$. We have V satisfies φ iff V' satisfies φ.*

Proof.

(i) Suppose V is feasible. We prove V' is also feasible by showing that $\gamma_{min} \leq |V'(i)| \leq \gamma_{max}$ for $i = 1, 2, \ldots, n$. We consider two cases according to whether $i \in CON$ or $i \in DIF$. The arguments for these two cases are in fact similar.

Fix an $i \in CON$. We have $\gamma_{min} \leq V(i) \leq \gamma_{max}$ or $-\gamma_{max} \leq V(i) \leq -\gamma_{min}$. Note that γ_{min}, γ_{max}, $-\gamma_{max}$, $-\gamma_{min}$ are integral multiples of Γ and are thus in Θ_{con}. Applying $\|V(i)\|_{con} = \|V'(i)\|_{con}$ then yields $\gamma_{min} \leq V'(i) \leq \gamma_{max}$ or $-\gamma_{max} \leq V'(i) \leq -\gamma_{min}$.

Fix an $i \in DIF$. We have $\gamma_{min} \leq V(i) \leq \gamma_{max}$ and thus $\ln\gamma_{min} \leq \ln V(i) \leq \ln\gamma_{max}$. Note that $\ln\gamma_{min}$, $\ln\gamma_{max}$ are members of Θ_{dif}. Applying $\|V(i)\|_{dif} = \|V'(i)\|_{dif}$ then yields $\ln\gamma_{min} \leq \ln V'(i) \leq \ln\gamma_{max}$ and hence $\gamma_{min} \leq |V'(i)| \leq \gamma_{max}$.

(ii) Suppose V satisfies φ. It suffices to show that for any constraint $x_i \prec c$ in φ, V satisfies $x_i \prec c$. To avoid repetition, we show this for $i \in DIF$. The argument for $i \in CON$ is similar but simpler.

Fix a constraint $x_i \prec c$ in φ where $i \in \mathit{DIF}$. Note that $V(i) \prec c$ and thus $\ln V(i) \prec \ln c$. Note that $\ln c$ is a member of Θ_{dif} and $\|V(i)\|_{\mathit{dif}} = \|V'(i)\|_{\mathit{dif}}$. It follows that $\ln V'(i) \prec \ln c$, that is, $V'(i) \prec c$.

$\square$

We are now ready show that $\approx$ is a congruence with respect to $\Longrightarrow_{\mathcal{A}}$.

Proposition 5.3. *Suppose* $(q1, V1, q1') \approx (q2, V2, q2')$ *and*

$$(q1, V1, q1') \overset{\alpha}{\Longrightarrow} (q3, V3, q3')$$

where $\alpha \in \{\tau, \mu\}$*. Then there exists a reachable configuration* $(q4, V4, q4')$ *such that*

$$(q2, V2, q2') \overset{\alpha}{\Longrightarrow} (q4, V4, q4')$$

and $(q3, V3, q3') \approx (q4, V4, q4')$.

Proof. Clearly $q1 = q2$ and $q1' = q2'$. We set $q4 = q3$ and $q4' = q3'$. Since $(q1, V1, q1')$ is feasible, $(q2, V2, q2')$ is also feasible by Observation 5.2(i). We show that exists a valuation $V4$ such that $(q2, V2, q2') \overset{\alpha}{\Longrightarrow}_{\mathcal{A}} (q4, V4, q4')$ and $\|V4\| = \|V3\|$. We consider two cases according to $\alpha = \tau$ or $\alpha = \mu$.

—**Case 1:** $\alpha = \tau$**.**

It follows from the definition of $\mathit{TS}_{\mathcal{A}}$ that there exist reals $t_i^{up} \in [\delta_{up}^0, \delta_{up}^1]$, $i = 1, 2, \ldots, n$, such that $\ln V3(i) = \ln V1(i) + A_{q'}(i,i) \cdot t_i^{up} + A_q(i,i) \cdot (1 - t_i^{up})$ for $i \in \mathit{DIF}$ and $V3(i) = V1(i) + b_{q'}(i) \cdot t_i^{up} + b_q(i) \cdot (1 - t_i^{up})$ for $i \in \mathit{CON}$. And $V3$ satisfies the invariant I_{q3}.

It suffices to show that there exist reals $s_i^{up} \in [\delta_{up}^0, \delta_{up}^1]$, $i = 1, 2, \ldots, n$, such that $\|V4\| = \|V3\|$, where $\ln V4(i) = \ln V2(i) + A_{q'}(i,i) \cdot s_i^{up} + A_q(i,i) \cdot (1 - s_i^{up})$ for $i \in \mathit{DIF}$ and $V4(i) = V2(i) + b_{q'}(i) \cdot s_i^{up} + b_q(i) \cdot (1 - s_i^{up})$ for $i \in \mathit{CON}$. Note that, by Observation 5.2, $\|V4\| = \|V3\|$ will guarantee that $V4$ satisfies the invariant I_{q4}, that is, I_{q3}.

In what follows, we will often need to give similar arguments for $i \in \mathit{DIF}$ and $i \in \mathit{CON}$. To avoid repetition, we will omit the latter.

Fix an $i \in \mathit{DIF}$. We show the existence of s_i^{up}. Assume $\|V3(i)\|_{dif} = (\theta, \theta')$ where $\theta, \theta' \in \Theta_{dif}$ and $A_{q'}(i,i) > A_q(i,i)$. It will become clear that other cases can be similarly handled. For any real u, let $\Phi^\tau(u)$ be the condition

$$\begin{array}{ll} \exists\, t^{up} \in \mathbb{R}. & \delta_{up}^0 \le t^{up} \le \delta_{up}^1 \\ \bigwedge & \theta < u + A_{q'}(i,i) \cdot t^{up} + A_q(i,i) \cdot (1 - t^{up}) < \theta' \ . \end{array}$$

It is easy to see that $\Phi^\tau(u)$ holds iff $\eta < u < \eta'$ where $\eta = \theta - A_{q'}(i,i) \cdot \delta_{up}^1 - A_q(i,i) \cdot (1 - \delta_{up}^1)$ and $\eta' = \theta' - A_{q'}(i,i) \cdot \delta_{up}^0 - A_q(i,i) \cdot (1 - \delta_{up}^0)$.

Since $\Phi^\tau(\ln V1(i))$ holds, we have $\eta < \ln V1(i) < \eta'$. Note that η, η' are members of Θ_{dif} (if $\eta, \eta' \in [\ln \gamma_{min}, \ln \gamma_{max}]$). Applying $\|V2(i)\|_{dif} = \|V1(i)\|_{dif}$ then yields $\eta < \ln V2(i) < \eta'$ and consequently $\Phi^\tau(\ln V2(i))$ holds. This establishes the existence of s_i^{up} for $i \in \mathit{DIF}$.

—**Case 2:** $\alpha = \mu$**.**

As in Case 1, it follows from the definition of $TS_{\mathcal{A}}$ that there exists a transition $(q1, g, q3)$ in $\longrightarrow$ and reals t_i^{up} in $[\delta_{up}^0, \delta_{up}^1]$, t_i^{ob} in $[\delta_{ob}^0, \delta_{ob}^1]$, $i = 1, 2, \ldots, n$ such that:

- The valuation U satisfies the guard g where $\ln U(i) = \ln V1(i) + A_{q'}(i,i) \cdot t_i^{up} + A_q(i,i) \cdot (t_i^{ob} - t_i^{up})$ for $i \in \mathit{DIF}$; $U(i) = V1(i) + b_{q'}(i) \cdot t_i^{up} + b_q(i) \cdot (t_i^{ob} - t_i^{up})$ for $i \in \mathit{CON}$.
- $\ln V3(i) = \ln V1(i) + A_{q'}(i,i) \cdot t_i^{up} + A_q(i,i) \cdot (1 - t_i^{up})$ for $i \in \mathit{DIF}$ and $V3(i) = V1(i) + b_{q'}(i) \cdot t_i^{up} + b_q(i) \cdot (1 - t_i^{up})$ for $i \in \mathit{CON}$.
- $V3$ satisfies the invariant I_{q3}.

We shall show the existence of reals $s_i^{up} \in [\delta_{up}^0, \delta_{up}^1]$, $s_i^{ob} \in [\delta_{ob}^0, \delta_{ob}^1]$, $i = 1, 2, \ldots, n$, such that:

- $\|U'\| = \|U\|$ where U' is the valuation given by $\ln U'(i) = \ln V2(i) + A_{q'}(i,i) \cdot s_i^{up} + A_q(i,i) \cdot (s_i^{ob} - s_i^{up})$ for $i \in \mathit{DIF}$ and $U'(i) = V2(i) + b_{q'}(i) \cdot s_i^{up} + b_q(i) \cdot (s_i^{ob} - s_i^{up})$ for $i \in \mathit{CON}$.

- $\|V4\| = \|V3\|$ where $V4$ is the valuation given by $\ln V4(i) = \ln V2(i) + A_{q'}(i,i) \cdot s_i^{up} + A_q(i,i) \cdot (1 - s_i^{up})$ for $i \in \mathit{DIF}$ and $V4(i) = V2(i) + b_{q'}(i) \cdot s_i^{up} + b_q(i) \cdot (1 - s_i^{up})$ for $i \in \mathit{CON}$.

By Observation 5.2(ii), $\|U'\| = \|U\|$ implies that U' satisfies the guard g, and $\|V4\| = \|V3\|$ guarantees $V4$ satisfies the invariant I_{q4}, that is I_{q3}. Hence the existence of s_i^{up}, s_i^{ob}, $i = 1, 2, \ldots, n$, suffices to complete the proof.

Fix an $i \in \mathit{DIF}$. Assume $\|V3(i)\|_{\mathit{dif}} = (\theta, \theta')$, $\|U(i)\|_{\mathit{dif}} = (\vartheta, \vartheta')$ where $\theta, \theta', \vartheta, \vartheta' \in \Theta_{\mathit{dif}}$ and $A_{q'}(i,i) > A_q(i,i) > 0$. It will become clear that other cases can be similarly handled. For any real u, let $\Phi^{\mu}(u)$ be the condition

$$\begin{array}{lll} \exists\, t^{up} \in \mathbb{R}.\, \exists\, t^{ob} \in \mathbb{R}. & & \delta_{up}^0 \le t^{up} \le \delta_{up}^1 \\ & \bigwedge & \theta < u + A_{q'}(i,i) \cdot t^{up} + A_q(i,i) \cdot (1 - t^{up}) < \theta' \\ & \bigwedge & \delta_{ob}^0 \le t^{ob} \le \delta_{ob}^1 \\ & \bigwedge & \vartheta < u + A_{q'}(i,i) \cdot t^{up} + A_q(i,i) \cdot (t^{ob} - t^{up}) < \vartheta' \; . \end{array}$$

As in Case 1, it is easy to see that $\Phi^{\mu}(u)$ holds iff $\eta < u < \eta'$, where η is the larger of $\theta - A_{q'}(i,i) \cdot \delta_{up}^1 - A_q(i,i) \cdot (1 - \delta_{up}^1)$ and $\vartheta - A_{q'}(i,i) \cdot \delta_{up}^1 - A_q(i,i) \cdot (\delta_{ob}^1 - \delta_{up}^1)$. On the other hand, η' is the smaller of $\theta' - A_{q'}(i,i) \cdot \delta_{up}^0 - A_q(i,i) \cdot (1 - \delta_{up}^0)$ and $\vartheta' - A_{q'}(i,i) \cdot \delta_{up}^0 - A_q(i,i) \cdot (\delta_{ob}^0 - \delta_{up}^0)$. It follows that η, η' are members of Θ_{dif} (if $\eta, \eta' \in [\ln \gamma_{min}, \ln \gamma_{max}]$). Thus, as in Case 1, one concludes that $\Phi^{\mu}(\ln V2(i))$ holds and the existence of s_i^{up}, s_i^{ob} for $i \in \mathit{DIF}$ is established.

By filling in similar but simpler arguments for $i \in \mathit{CON}$, we can complete the proof of proposition 5.3. $\square$

Having established the claim that $\approx$ is a congruence with respect to $\Longrightarrow_{\mathcal{A}}$, we are now ready to prove theorem 5.1.

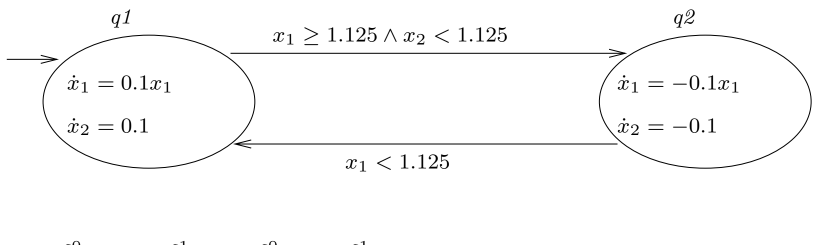
Proof of theorem 5.1. Clearly, the members of Θ_{dif} and Θ_{con} can be effectively represented. Further, the members of Θ_{dif} (Θ_{con}) can be effectively ordered and thus the finitely many equivalence classes of $\approx$ can be effectively represented. Note that, to compare two members of Θ_{dif} one just needs to determine whether $e^{m_1} < m_2$ for integers m_1, m_2. This can be done by approximating e sufficiently precisely using for instance the power series expansion of e. In fact, we note that for any polynomial $f(u)$ in one variable u with integer coefficients, we can effectively whether $f(u) < 0$. Since $e = 1 + \sum_{h=1}^{\infty} 1/h!$, we have

$$1 + \sum_{h=1}^{\ell} \frac{1}{h!} \; < \; e \; < \; 1 + \sum_{h=1}^{\ell} \frac{1}{h!} + \sum_{h=\ell+1}^{\infty} \frac{1}{\ell^{h-\ell}} \; = \; 1 + \sum_{h=1}^{\ell} \frac{1}{h!} + \frac{1}{\ell - 1} \; .$$

Note that the polynomial $f(u)$ has finitely many real roots. Hence for sufficiently large ℓ, $f(u)$ has no root in the interval $[1 + \sum_{h=1}^{\ell} 1/h!,$ $1+\sum_{h=1}^{\ell} 1/h!+1/(\ell-1)]$ and so $f(e)$ has the same sign as $f(1+\sum_{h=1}^{\ell} 1/h!)$. Clearly such an ℓ can be effectively found.

We shall refer to equivalence classes of $\approx$ as *clusters* of $\mathcal{A}$. We denote the cluster containing (q, V, q') by $[(q, V, q')]_{\approx}$, or simply $[(q, V, q')]$. Now construct a finite transition system $CA_{\mathcal{A}} = (CL_{\mathcal{A}}, [(q_{in}, V_{in}, q_{in})], \{\tau, \mu\}, \rightsquigarrow_{\mathcal{A}})$ which we called the *cluster automaton* of $\mathcal{A}$. We have that $CL_{\mathcal{A}}$ is the *finite* set of clusters of $\mathcal{A}$. The transition relation $\rightsquigarrow_{\mathcal{A}}$ is a subset of $CL_{\mathcal{A}} \times \{\tau, \mu\} \times CL_{\mathcal{A}}$ and is given by: there is a transition from $C1 \overset{\alpha}{\rightsquigarrow}_{\mathcal{A}} C2$ iff there exists (q, V, q') in $C1$, $(q1, V1, q1')$ in $C2$ such that $(q, V, q') \overset{\alpha}{\Longrightarrow}_{\mathcal{A}} (q1, V1, q1')$.

From the proof of proposition 5.3, to determine whether there exists a transition $C1 \overset{\alpha}{\rightsquigarrow}_{\mathcal{A}} C2$ amounts to comparing members of Θ_{dif} (and Θ_{con}). Hence the transition system $CA_{\mathcal{A}}$ can be effectively computed from $\mathcal{A}$. It is now straightforward to construct from $CA_{\mathcal{A}}$ a finite state automaton which accepts $L_{st}(\mathcal{A})$. This completes the proof of theorem 5.1. □



$\delta^0_{ob} = .75,\ \delta^1_{ob} = 1,\ \delta^0_{up} = 0,\ \delta^1_{up} = .25$
$V_{in}(1) = V_{in}(2) = 1.025$
$\gamma_{min} = 1.025,\ \gamma_{max} = 1.15$

FIGURE 5.4. A simple RDA

For illustration, we construct the cluster automaton of the RDA shown in figure 5.4. Both invariants of $q1$ and $q2$ are true. We have $\Delta = .25$ and thus $\Gamma = .025$. Elements of the set Θ_{con} are shown in figure 5.5. Note that $\gamma_{min} = 41\Gamma$ and $\gamma_{max} = 46\Gamma$.

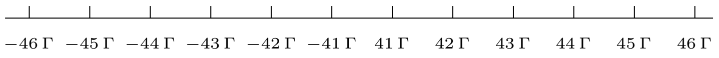


FIGURE 5.5. The set Θ_{con} (not to scale)

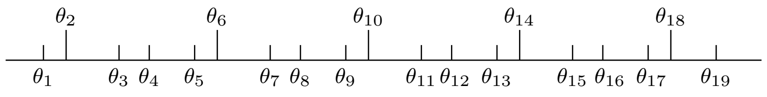


FIGURE 5.6. The set Θ_{dif} (not to scale)

The set Θ_{IR} consists of the reals $\ln 1.025$, $\ln 1.15$, $\ln 1.125$, which are approximately equal to $.99\Gamma$, 5.59Γ, 4.71Γ, respectively. Elements of the set Θ_{dif} are shown in figure 5.6, where

$$\begin{array}{ll}
\theta_1 = \ln 1.025, & \theta_2 = \Gamma, \\
\theta_3 = -4\Gamma + \ln 1.15, & \theta_4 = -3\Gamma + \ln 1.125, \\
\theta_5 = \Gamma + \ln 1.025, & \theta_6 = 2\Gamma, \\
\theta_7 = -3\Gamma + \ln 1.15, & \theta_8 = -2\Gamma + \ln 1.125, \\
\theta_9 = 2\Gamma + \ln 1.025, & \theta_{10} = 3\Gamma, \\
\theta_{11} = -2\Gamma + \ln 1.15, & \theta_{12} = -\Gamma + \ln 1.125, \\
\theta_{13} = 3\Gamma + \ln 1.025, & \theta_{14} = 4\Gamma, \\
\theta_{15} = -\Gamma + \ln 1.15, & \theta_{16} = \ln 1.125, \\
\theta_{17} = 4\Gamma + \ln 1.025, & \theta_{18} = 5\Gamma, \\
\theta_{19} = \ln 1.15\ . &
\end{array}$$

It follows that each cluster is (q, J_1, J_2, q') where $q, q' \in \{q1, q2\}$ and J_1 is a member of $\mathcal{I}_{dif}$, J_2 is a member of $\mathcal{I}_{con}$. In figure 5.7, we display a fragment of the cluster automaton, where

$$C1 = (q1,\ [\theta_1, \theta_1],\ [41\Gamma,\ 41\Gamma],\ q1),$$

$$C2 = (q1,\ [\theta_{17}, \theta_{17}],\ [45\Gamma,\ 45\Gamma],\ q1),$$

$$C3 = (q1,\ (\theta_{19}, \infty),\ (46\Gamma, \infty),\ q1),$$

$$C4 = (q2,\ [\theta_{17}, \theta_{17}],\ [45\Gamma,\ 45\Gamma],\ q1)\ .$$

And to reduce clutter, the dotted arrow from $C4$ to C_{ij} with label τ represents the collection of transitions from $C4$ to each $C_{ij} = (q2, (J_i^1, J_j^2), q2)$ with label τ, where J_i^1 is an interval in

$$\{[\theta_h, \theta_h] \mid h = 1, 2, \ldots, 9\}$$
$$\bigcup \{(\theta_h, \theta_{h+1}) \mid h = 1, 2, \ldots, 8\}$$

and J_j^2 is an interval in

$$\{[h\Gamma,\, h\Gamma] \mid h = 41, 42, 43\}$$
$$\bigcup \{(h\Gamma,\, (h+1)\Gamma) \mid h = 41, 42\}\ .$$

Similarly the dotted arrow from $C4$ to C'_{ij} with label μ represents the collection of transitions from $C4$ to each $C'_{ij} = (q1,\, (J_i^1, J_j^2),\, q2)$ with label μ, where J_i^1, J_j^2 are as described above.

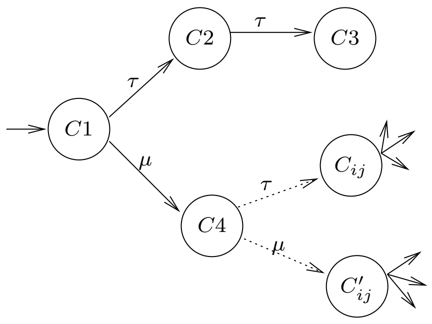


FIGURE 5.7. Cluster automaton

5.5. Controller Synthesis for RDAs

We now define the controller synthesis problem associated with RDAs. We shall view the RDA $\mathcal{A}$ naturally as a *plant*, which describes the possible interactions between a system interacting with a continuous environment. At each time instant T_k, the system can decide whether to stay at the current control state q provided the associated invariant I_q is satisfied, or to switch to another control state q' provided the associated guard and $I_{q'}$ is satisfied. The goal is to derive a strategy which advises the moves of the system at each time instant T_k such that the controlled behaviour of plant, in terms of the state sequences, satisfies a specification.

Through the rest of this chapter, we assume that $\delta^0_{ob} = 1 = \delta^1_{ob}$ and tackle the controller synthesis problem in this setting. Hence the values of the x_i's that are observed at time instant T_k are equal to the actual values of the x_i's at T_k. We will also assume from now on that $\mathcal{A}$ is augmented with a set AP of atomic propositions and a labelling function $\lambda_{\mathcal{A}} : Q \to 2^{AP}$.

We shall consider linear time specifications given as LTL or QPLTL formulae over AP. Let ψ be an LTL formula over AP. For an infinite run $\sigma = (q_0, V_0, q'_0)(q_1, V_1, q'_1) \ldots$ of $\mathcal{A}$. We say σ is a model of ψ iff $\lambda(q_0)\lambda(q_1)\ldots$ is a model of ψ.

A *strategy* f for $\mathcal{A}$ is a function $\mathit{Runs}(\mathcal{A}) \to 2^Q$ which satisfies the following: Suppose $\sigma = (q_0, V_0, q'_0)\,(q_1, V_1, q'_1)\,\ldots\,(q_\ell, V_\ell, q'_\ell)$ is a run in $\mathit{Runs}(\mathcal{A})$. Then for each q in $f(\sigma)$, there exists a reachable configuration $(\hat{q}, \widehat{V}, \hat{q}')$ such that $\hat{q} = q$ and $(q_\ell, V_\ell, q'_\ell) \overset{\alpha}{\Longrightarrow} (\hat{q}, \widehat{V}, \hat{q}')$ for some $\alpha \in \{\tau, \mu\}$. Thus the strategy f recommends only structurally possible moves.

The set of runs according to the strategy f, denoted $\mathit{Runs}(f)$, is defined inductively as follows:

- ε is in $\mathit{Runs}(f)$.
- If $\sigma = (q_0, V_0, q'_0)\,(q_1, V_1, q'_1)\,\ldots(q_\ell, V_\ell, q'_\ell)$ is in $\mathit{Runs}(f)$ and $\sigma' = \sigma\,(\hat{q}, \widehat{V}, \hat{q}')$ with $\hat{q} \in f(\sigma)$, then $\sigma' \in \mathit{Runs}(f)$.

Infinite runs of $\mathcal{A}$ and infinite runs according to f are defined in the obvious way. Namely, $\sigma \in RC^\omega$ is an infinite run of $\mathcal{A}$ iff every finite prefix of σ is in $\mathit{Runs}(\mathcal{A})$. An infinite run σ is according to f iff every finite prefix of σ is according to f. We say the strategy f for $\mathcal{A}$ is *non-blocking* iff every run according to f can be extended to a longer run according to f. In particular, this implies that every run according to f will not reach any infeasible configurations.

We say the strategy f is *ψ-winning* iff f is non-blocking and every infinite run according to f is a model of ψ. By a *ψ-controller* for $\mathcal{A}$, we shall mean a ψ-winning strategy for $\mathcal{A}$.

5.6. Decidability Results

Our result for controller synthesis of RDAs is:

Theorem 5.4.

(i) *Given a pair $(\mathcal{A}, \psi)$, where $\mathcal{A}$ is an RDA with $\delta^0_{ob} = 1 = \delta^1_{ob}$ and ψ is an LTL formula over AP, we can effectively determine whether there exists a ψ-controller for $\mathcal{A}$.*

(ii) *Further, if there exists a ψ-controller for $\mathcal{A}$, then we can effectively construct one in the form of a (finite) RDA with no sensing delay.*

In the rest of this section we prove theorem 5.4(i) and analyze the complexity of the decision procedure. We then prove theorem 5.4(ii). We also show that theorem 5.4 can be easily extended to QPLTL specifications. The proofs of these results are technically similar to the proofs of theorem 4.2 and 4.7.

Recall the cluster automaton of $\mathcal{A}$ from the proof of theorem 5.1. With $CA_{\mathcal{A}}$, we show that there exists a ψ-controller iff there exists a *cluster-respecting* ψ-controller (one that does not distinguish two runs that pass through the same sequence of clusters). This result is the key for establishing theorem 5.4. With this result in mind, we show how to determine the existence of cluster-respecting ψ-controllers by constructing a non-deterministic Rabin tree automaton $\mathcal{R}_{ctrl}$ which will run over $\{\top, \bot\}$-labelled trees whose underlying tree is the computation tree induced by the cluster automaton of $\mathcal{A}$. And $\mathcal{R}_{ctrl}$ will accepts a $\{\top, \bot\}$-labelled tree iff this labelled tree represents a cluster-respecting ψ-controller. This will settle the first part of theorem 5.4, namely, the decidability problem. Further, due to Rabin's tree theorem [63], in case the set of labelled trees accepted by $\mathcal{R}_{ctrl}$ is nonempty then it in fact accepts a *regular* $\{\top, \bot\}$-labelled tree. This regular tree can be effectively computed, represented as a finite structure $\mathcal{C}$ and this structure can be naturally viewed as an RDA with no sensing delay. This RDA will then constitute the controller we seek.

5.6.1. Cluster-Respecting Strategies.

In order to prove theorem 5.4(i), we first show a crucial lemma which allows us to deal with only *cluster-respecting* strategies. Intuitively, a cluster-respecting strategy is one that does not discriminate between two histories of configurations that pass through the same sequence of clusters.

We fix ψ (and $\mathcal{A}$) as stated in theorem 5.4. We shall assume for notational convenience the only deadlocked configurations in $TS_{\mathcal{A}}$ are infeasible configurations. In other words, for every reachable configuration (q, V, q') in RC, if (q, V, q') is feasible, then there exists $(q1, V1, q1')$ in RC such that $(q, V, q') \overset{\alpha}{\Longrightarrow} (q1, V1, q1')$.

We extend the equivalence relation $\approx$ on reachable configurations to runs of $\mathcal{A}$ in the obvious way. Namely, if $\sigma = (q_0, V_0, q_0')\,(q_1, V_1, q_1')\ \ldots\ (q_\ell, V_\ell, q_\ell')$ and $\hat{\sigma} = (\hat{q}_0, \widehat{V}_0, \hat{q}_0')\,(\hat{q}_1, \widehat{V}_1, \hat{q}_1')\ \ldots\ (\hat{q}_\ell, \widehat{V}_\ell, \hat{q}_\ell')$ are runs in $\mathcal{A}$, then we say $\sigma \approx \sigma'$ iff $(q_i, V_i, q_i') \approx (\hat{q}_i, \widehat{V}_i, \hat{q}_i')$ for $i = 1, 2, \ldots, \ell$.

Let f be a strategy for $\mathcal{A}$. We say f is *cluster-respecting* iff the following holds: for every σ, σ' in $Runs(f)$, if $\sigma \approx \sigma'$, then $f(\sigma) = f(\sigma')$. The lemma below is the key for establishing theorem 5.4.

The following lemma is crucial for establishing theorem 5.4.

Lemma 5.5. *Let ψ be an LTL specification. Then there exists a ψ-controller for $\mathcal{A}$ iff there exists a cluster-respecting ψ-controller for $\mathcal{A}$.*

Proof. Suppose f is a ψ-controller. We shall construct a cluster-respecting ψ-controller from f.

We begin by constructing REP_f, a "representative" prefix-closed subset of $Runs(f)$ inductively as follows.

Firstly, ε is in REP_f and also the run (q_{in}, V_{in}, q_{in}) is in REP_f.

Secondly, suppose $\sigma = (q_0, V_0, q_0')\,(q_1, V_1, q_1')\,\ldots\,(q_\ell, V_\ell, q_\ell')$ is in *Runs*(f). Call Z in *CL* a *successor cluster* of σ iff $(q_\ell, V_\ell, q_\ell') \overset{\alpha}{\Longrightarrow} (\hat{q}, \widehat{V}, \hat{q}')$ for some $\alpha \in \{\tau, \mu\}$ and $(\hat{q}, \widehat{V}, \hat{q}')$ in Z. Let $Z_1, Z_2, \ldots, Z_m$ be the set of successor clusters of σ. Note that since f is non-blocking, $(q_\ell, V_\ell, q_\ell')$ is a feasible configuration and hence $Z_1, Z_2, \ldots, Z_m$ exist.

We pick $(\hat{q}_1, \widehat{V}_1, \hat{q}_1')$ in Z_1, $(\hat{q}_2, \widehat{V}_2, \hat{q}_2')$ in Z_2, $\ldots$, $(\hat{q}_m, \widehat{V}_m, \hat{q}_m')$ in Z_m such that $(q_\ell, V_\ell, q_\ell') \overset{\alpha_j}{\Longrightarrow} (\hat{q}_j, \widehat{V}_j, \hat{q}_j')$, for $j = 1, 2, \ldots, m$, where $\alpha_j \in \{\tau, \mu\}$. Now we let $\sigma\ (\hat{q}_j, \widehat{V}_j, \hat{q}_j') \in REP_f$, for $i = 1, 2, \ldots, m$.

We argue that the choices of $(\hat{q}_j, \widehat{V}_j, \hat{q}_j')$, $j = 1, 2, \ldots, m$, can be made effective and hence we need *not* appeal to the axiom of choice ([34]) here.

Fix $j \in \{1, 2, \ldots, m\}$. Let $\left\|\widehat{V}_j\right\| = (L_1, L_2, \ldots, L_n)$. Now to pick $(\hat{q}_j, \widehat{V}_j, \hat{q}_j')$ effectively, it suffices to choose effectively n real numbers t_i^{up} in $[\delta_{up}^0, \delta_{up}^1]$, $i = 1, 2, \ldots, n$, such that $V_\ell(i) \cdot \exp\bigl(A_{q_\ell'}(i,i) \cdot t_i^{up} + A_{q_\ell}(i,i) \cdot (1 - t_i^{up})\bigr)$ is in L_i for $i \in DIF$ and $V_\ell(i) + b_{q_\ell'}(i) \cdot t_i^{up} + b_{q_\ell}(i) \cdot (1 - t_i^{up})$ is in L_i for $i \in CON$.

Fix $i \in DIF$. We show that t_i^{up} can be picked effectively.

Assume that $L_i = (\theta, \theta')$ where $\theta, \theta' \in \Theta_{dif}$ and $A_{q_\ell'}(i,i) > A_{q_\ell}(i,i)$. It will become clear that other cases can be similarly handled.

We pick t_i^{up} as follows. If $\vartheta < \delta_{up}^1$ where $\vartheta = \bigl(\ln\theta - \ln V_\ell(i) - A_{q_\ell}(i,i)\bigr) \cdot (A_{q_\ell'} - A_{q_\ell})^{-1}$, then we pick $t_i^{up} = 1/2(\vartheta + \delta_{up}^1)$. Otherwise we have $\delta_{up}^0 < \vartheta'$, where $\vartheta' = \bigl(\ln\theta' - \ln V_\ell(i) - A_{q_\ell}(i,i)\bigr) \cdot (A_{q_\ell'} - A_{q_\ell})^{-1}$, In this case, we pick $t_i^{up} = 1/2(\delta_{up}^0 + \vartheta')$.

By filling similar but simpler arguments for the effective choices of t_i^{up} for $i \in CON$, we establish the claim that t_i^{up}, $i = 1, 2, \ldots, m$, can be made effectively.

Now we construct the cluster-respecting strategy $\hat{f}$ from REP_f as follows. For σ in *Runs*$(\mathcal{A})$, if there exists σ' in REP_f such that $\sigma \approx \sigma'$, then $\hat{f}(\sigma) = f(\sigma')$; otherwise $\hat{f}(\sigma) = \emptyset$. It is now routine to show that $\hat{f}$ is in fact a well-defined cluster-respecting ψ-controller. $\square$

5.6.2. Decidability for LTL Specifications.

With lemma 5.5, to determine whether there exists a ψ-controller for $\mathcal{A}$, one just need to determine whether there exists a cluster-respecting ψ-controller for $\mathcal{A}$. We next show that the latter can be done effectively. This will settle the first part of theorem 5.4.

In what follows, we fix a two letter alphabet $\{\top, \bot\}$, where $\top$, $\bot$ are special symbols. We show that cluster-respecting strategies can be represented as $\{\top, \bot\}$-labelled trees and one can effectively construct a non-deterministic Rabin tree automaton which accepts the set of $\{\top, \bot\}$-labelled trees representing cluster-respecting ψ-winning strategies.

First we construct a *CL*-tree $\mathcal{T}$ by unfolding *CA*, the cluster automaton of $\mathcal{A}$ with special handling on clusters containing infeasible configurations. For a cluster $[(q, V, q')]$ in *CL*, we define the set $Succ_{CA}([(q, V, q')])$ by: $[(\hat{q}, \widehat{V}, \hat{q}')] \in CL$ is in $Succ_{CA}([(q, V, q')])$ iff $[(q, V, q')] \overset{\alpha}{\rightsquigarrow} [(\hat{q}, \widehat{V}, \hat{q}')]$ for some $\alpha \in \{\tau, \mu\}$. Formally, we define the *CL*-tree $\mathcal{T}$ inductively as follows.

- ε is in $\mathcal{T}$ and $[(q_{in}, V_{in}, q_{in})]$ is in $\mathcal{T}$.
- Suppose $\sigma = [(q_0, V_0, q_0')]\ [(q_1, V_1, q_1')]\ \ldots\ [(q_\ell, V_\ell, q_\ell')]$ is in $\mathcal{T}$ where $[(q_i, V_i, q_i')] \in CL$ for $i = 0, 1, \ldots, \ell$.
  - If $(q_\ell, V_\ell, q_\ell')$ is a feasible configuration (see proposition 5.3), then for each $[(\hat{q}, \widehat{V}, \hat{q}')]$ in $Succ_{CA}([(q_\ell, V_\ell, q_\ell')])$, we let $\sigma\ [(\hat{q}, \widehat{V}, \hat{q}')]$ be in $\mathcal{T}$.
  - If $(q_\ell, V_\ell, q_\ell')$ is an infeasible configuration, then we let $\sigma\ [(q_\ell, V_\ell, q_\ell')]$ be in $\mathcal{T}$.

Clearly $\mathcal{T}$ is a *CL*-tree and for every σ in $\mathcal{T}$, $Succ_{\mathcal{T}}(\sigma) \neq \emptyset$.

Let $(\mathcal{T}, \eta)$ be a $\{\top, \bot\}$-labelled *CL*-tree, where $\eta : \mathcal{T} \to \{\top, \bot\}$ is a labelling function. We say $(\mathcal{T}, \eta)$ is a *strategy tree* iff the following hold:

- $\eta(\varepsilon) = \top$ and $\eta([(q_{in}, V_{in}, q_{in})]) = \top$.
- Suppose $\sigma = [(q_0, V_0, q_0')]\ [(q_1, V_1, q_1')]\ \ldots\ [(q_\ell, V_\ell, q_\ell')]$ is in $\mathcal{T}$ and $\eta(\sigma) = \top$.

  - Suppose $(q_\ell, V_\ell, q'_\ell)$ is a feasible configuration. Then for any $[(\hat{q}_1, \widehat{V}_1, \hat{q}'_1)]$, $[(\hat{q}_2, \widehat{V}_2, \hat{q}'_2)]$ in CL such that $\sigma\ [(\hat{q}_1, \widehat{V}_1, \hat{q}'_1)]$ and $\sigma\ [(\hat{q}_2, \widehat{V}_2, \hat{q}'_2)]$ are both in $\mathcal{T}$, $\hat{q}_1 = \hat{q}_2$ implies that $\eta(\sigma\ [(\hat{q}_1, \widehat{V}_1, \hat{q}'_1)]) = \eta(\sigma\ [(\hat{q}_2, \widehat{V}_2, \hat{q}'_2)])$.
  - Suppose $(q_\ell, V_\ell, q'_\ell)$ is an infeasible configuration. Then for any $[(\hat{q}, \widehat{V}, \hat{q}')]$ in CL such that $\sigma\ [(\hat{q}, \widehat{V}, \hat{q}')]$ is in $\mathcal{T}$, we have $\eta(\sigma\ [(\hat{q}, \widehat{V}, \hat{q}')]) = \bot$.

- Suppose $\sigma = [(q_0, V_0, q'_0)]\ [(q_1, V_1, q'_1)]\ \ldots\ [(q_\ell, V_\ell, q'_\ell)]$ is in $\mathcal{T}$ and $\eta(\sigma) = \bot$. Then for any $[(\hat{q}, \widehat{V}, \hat{q}')]$ in CL such that $\sigma\ [(\hat{q}, \widehat{V}, \hat{q}')]$ is in $\mathcal{T}$, we have $\eta(\sigma\ [(\hat{q}, \widehat{V}, \hat{q}')]) = \bot$.

It is easy to see that there is a 1-1 correspondence between cluster-respecting strategies and strategy trees. In fact, if f is a cluster-respecting strategy, then one can define the $\{\top, \bot\}$-labelled $CL_{\mathcal{A}}$-tree $(\mathcal{T}, \eta_f)$ as follows: for $\sigma = [(q_0, V_0, q'_0)]\ [(q_1, V_1, q'_1)]\ \ldots\ [(q_\ell, V_\ell, q'_\ell)]$ in $\mathcal{T}$, we have $\eta_f(\sigma) = \top$ if there exists $\hat{\sigma}$ in $Runs(f)$ where $\hat{\sigma} = (\hat{q}_0, \widehat{V}_0, \hat{q}'_0)\,(\hat{q}_1, \widehat{V}_1, \hat{q}'_1)\ \ldots\ (\hat{q}_\ell, \widehat{V}_\ell, \hat{q}'_\ell)$ with $(\hat{q}_j, \widehat{V}_j, \hat{q}'_j)$ in $[(q_j, V_j, q'_j)]$ for $j = 1, 2, \ldots, \ell$. And $\eta_f(\sigma) = \bot$ otherwise. It is clear that $(\mathcal{T}, \eta_f)$ is a strategy tree. On the other hand, suppose $(\mathcal{T}, \eta)$ is a strategy tree, then one can define a cluster-respecting strategy f_η as follows: for $\sigma = (q_0, V_0, q'_0)\,(q_1, V_1, q'_1)\ \ldots\ (q_\ell, V_\ell, q'_\ell)$ in $Runs(\mathcal{A})$, $f_\eta(\sigma)$ is given by: q in Q is in $f_\eta(\sigma)$ iff there exists $[(\hat{q}, \widehat{V}, \hat{q}')]$ in CL such that $\hat{q} = q$ and $\eta(\sigma\ [(\hat{q}, \widehat{V}, \hat{q}')]) = \top$. It is straightforward to verify that f_η is well-defined strategy and is cluster-respecting.

Now it is routine to show that a strategy tree $(\mathcal{T}, \eta)$ represents a cluster-respecting ψ-winning strategy iff $(\mathcal{T}, \eta)$ satisfies the following conditions:

- (non-blocking) For every σ in $\mathcal{T}$ with $\eta(\sigma) = \top$, there exists $[(q, V, q')]$ in CL such that $\sigma[(q, V, q')]$ is in $\mathcal{T}$ and $\eta(\sigma[(q, V, q')]) = \top$.
- Suppose $[(q_0, V_0, q'_0)]\ [(q_1, V_1, q'_1)]\ \ldots$ is a path in $\mathcal{T}$. If for every $\ell = 0, 1, \ldots,$ the node $[(q_0, V_0, q'_0)]\ [(q_1, V_1, q'_1)]\ \ldots\ [(q_\ell, V_\ell, q'_\ell)]$ in $\mathcal{T}$ is labelled $\top$ by η, then $(q_0, V_0, q'_0)\,(q_1, V_1, q'_1)\ \ldots$ is a model of ψ.

We can now construct a non-deterministic Rabin tree automaton $\mathcal{R}_{ctrl}$ which will run over $\{\top, \bot\}$-labelled CL-trees (whose underlying tree is $\mathcal{T}$) such that $\mathcal{R}_{ctrl}$ accepts $(\mathcal{T}, \eta)$ iff $(\mathcal{T}, \eta)$ is a strategy tree representing a ψ-winning cluster-respecting strategy. It will be convenient to view $\mathcal{R}_{ctrl}$ as the intersection of three non-deterministic tree automata $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{R}_3$, where $\mathcal{B}_1$, $\mathcal{B}_2$ are non-deterministic Büchi tree automata and $\mathcal{R}_3$ is a deterministic Rabin tree automaton. For an input $\{\top, \bot\}$-labelled CL-tree $(\mathcal{T}, \eta)$, $\mathcal{B}_1$ checks whether $(\mathcal{T}, \eta)$ is a strategy tree, $\mathcal{B}_2$ examines whether the cluster-respecting strategy represented by $(\mathcal{T}, \eta)$ is non-blocking, and $\mathcal{R}_3$ verifies that for every path σ of $\mathcal{T}$ along which every node is labelled $\top$, σ is a model of ψ (in the sense defined above).

We have $\mathcal{B}_1 = ((CL \cup \{\$\}) \times \{\top, \bot\}, (\$, \top), \{\top, \bot\}, \hookrightarrow_1, CL \times \{\top, \bot\})$ where \$ is a special symbol and $\hookrightarrow_1$ is given by:

- $(\$, \top) \overset{\top}{\hookrightarrow}_1 \chi$, where $\chi : \{[(q_{in}, V_{in}, q'_{in})]\} \to (CL \times \{\top, \bot\})$ maps $[(q_{in}, V_{in}, q'_{in})]$ to $([(q_{in}, V_{in}, q'_{in})], \top)$.
- Suppose $[(q, V, q')]$ is in CL where (q, V, q') is a feasible configuration. Let $Succ_{CA}([(q, V, q')]) = \{Z_1, Z_2, \ldots, Z_m\}$. Then we have $([(q, V, q')], \top) \overset{\top}{\hookrightarrow}_1 \chi$ for any $\chi : Succ_{CA}([(q, V, q')]) \to (CL \times \{\top, \bot\})$ which maps Z_i to (Z_i, b_i) where $b_i \in \{\top, \bot\}$, for each $i = 1, 2, \ldots, m$ and satisfies the following condition: for any $Z_i = [(\hat{q}_i, \widehat{V}_i, \hat{q}'_i)]$, $Z_j = [(\hat{q}_j, \widehat{V}_j, \hat{q}'_j)]$ with $\hat{q}_i = \hat{q}_j$, we have $b_i = b_j$.

  Further, $([(q, V, q')], \bot) \overset{\bot}{\hookrightarrow}_1 \chi$ where $\chi : Succ_{CA}([(q, V, q')]) \to (CL \times \{\top, \bot\})$ maps Z_i to $(Z_i, \bot)$ for each $i = 1, 2, \ldots, m$.
- Suppose $[(q, V, q')]$ is in CL where (q, V, q') is an infeasible configuration. Then $([(q, V, q')], \top) \overset{\top}{\hookrightarrow}_1 \chi$ and $([(q, V, q')], \bot) \overset{\bot}{\hookrightarrow}_1 \chi$, where $\chi : \{[(q, V, q')]\} \to CL \times \{\top, \bot\}$ maps $[(q, V, q')]$ to $([(q, V, q')], \bot)$.

Intuitively, for an input $\{\top, \bot\}$-labelled *CL*-tree $(\mathcal{T}, \eta)$, a state $([(q, V, q')], b)$ of $\mathcal{B}_1$ indicates that $\mathcal{B}_1$ expects a node σ in $\mathcal{T}$ such that $dir(v) = [(q, V, q')]$ and $\eta(v) = b$. It is straightforward to verify that $\mathcal{B}_1$ accepts $(\mathcal{T}, \eta)$ iff $(\mathcal{T}, \eta)$ is a strategy tree.

We next define $\mathcal{B}_2 = ((CL \cup \{\$\}) \times \{\top, \bot\}, (\$, \top), \{\top, \bot\}, \hookrightarrow_2, CL \times \{\top, \bot\})$ where \$ is a special symbol and $\hookrightarrow_2$ is given by:

- $(\$, \top) \overset{\top}{\hookrightarrow}_2 \chi$ where $\chi : \{[(q_{in}, V_{in}, q'_{in})]\} \rightarrow (CL \times \{\top, \bot\})$ maps $[(q_{in}, V_{in}, q'_{in})]$ to $([(q_{in}, V_{in}, q'_{in})], \top)$.
- Suppose $[(q, V, q')]$ is in *CL* where (q, V, q') is a feasible configuration. Let $Succ_{CA}([(q, V, q')]) = \{Z_1, Z_2, \ldots, Z_m\}$. Then $([(q, V, q')], \top) \overset{\top}{\hookrightarrow}_2 \chi$ for any function χ from $Succ_{CA}([(q, V, q')])$ to $CL \times \{\top, \bot\}$ which maps each Z_i to (Z_i, b_i), where $b_i \in \{\top, \bot\}$ for $i = 1, 2, \ldots, m$, and satisfies the following condition: there exists at least one i in $\{1, 2, \ldots, m\}$ with $b_i = \top$.

  Further, $([(q, V, q')], \bot) \overset{\bot}{\hookrightarrow}_2 \chi$ where $\chi : Succ_{CA}([(q, V, q')]) \rightarrow (CL \times \{\top, \bot\})$ maps each Z_i to $(Z_i, \bot)$ for $i = 1, 2, \ldots, m$.
- Suppose $[(q, V, q')]$ is in *CL* where (q, V, q') is an infeasible configuration. Then $([(q, V, q')], \bot) \overset{\bot}{\hookrightarrow}_2 \chi$, where $\chi : \{[(q, V, q')]\} \rightarrow CL \times \{\top, \bot\}$ maps $[(q, V, q')]$ to $([(q, V, q')], \bot)$.

Similarly to $\mathcal{B}_1$, for an input $\{\top, \bot\}$-labelled *CL*-tree $(\mathcal{T}, \eta)$, a state $([(q, V, q')], b)$ of $\mathcal{B}_2$ indicates that $\mathcal{B}_2$ expects a node σ in $\mathcal{T}$ such that $dir(\sigma) = [(q, V, q')]$ and $\eta(v) = b$. It is again clear that $\mathcal{B}_2$ accepts a strategy tree $(\mathcal{T}, \eta)$ iff the strategy represented by $(\mathcal{T}, \eta)$ is non-blocking.

To define $\mathcal{R}_3$, we first note that for the LTL formula ψ, one can effectively construct a non-deterministic Büchi automaton $\mathcal{B}_\psi$ over 2^{AP} ([78]) which has the following property: Let σ be in $(2^{AP})^\omega$. Then σ is accepted by $\mathcal{B}_\psi$ iff σ is a model of ψ. From $\mathcal{B}_\psi$, we can construct a *deterministic* Rabin automaton $\mathcal{R}_\psi$ ([68]) over 2^{AP} such that: for every σ in $(2^{AP})^\omega$, σ is accepted by $\mathcal{B}_\psi$ iff σ is accepted by $\mathcal{R}_\psi$. The Rabin tree automaton $\mathcal{R}_3$ will simulate $\mathcal{R}_\psi$ along every path of which every node is labelled $\top$. The crucial point to note is that $\mathcal{R}_\psi$ is deterministic.

Suppose $\mathcal{R}_\psi = (S_\psi, s^\psi_{in}, 2^{AP}, \hookrightarrow_\psi, \mathcal{F}_\psi)$, where $\mathcal{F}_\psi = \{(E_1, F_1), (E_2, F_2), \dots, (E_k, F_k)\}$. We define $\mathcal{R}_3 = (CL \times (S_\psi \cup \{\sharp\}), (\$, s^\psi_{in}), \{\top, \bot\}, \hookrightarrow_3, \widehat{\mathcal{F}})$ with $\widehat{\mathcal{F}} = \{\big(CL \times E_i, CL \times (F_i \cup \{\sharp\})\big) \mid i = 1, 2, \dots, k\}$, where $\sharp$ is a special symbol and is not in S_ψ. And $\hookrightarrow_3$ is defined as follows:

- $(\$, s^\psi_{in}) \overset{\top}{\hookrightarrow}_3 \chi$, where $\chi : \{[(q_{in}, V_{in}, q'_{in})]\} \to (CL \times S_\psi)$ maps $[(q_{in}, V_{in}, q'_{in})]$ to $([(q_{in}, V_{in}, q'_{in})], s^\psi_{in})$.
- Suppose $[(q, V, q')]$ is in CL where (q, V, q') is a feasible configuration. Let $\mathit{Succ}_{CA}([(q, V, q')]) = \{Z_1, Z_2, \dots, Z_m\}$.

  For $s \in S_\psi$, we have:

  - $([(q, V, q')], s) \overset{\top}{\hookrightarrow}_3 \chi$ where $\chi : \mathit{Succ}_{CA}([(q, V, q')]) \to (CL \times S_\psi)$ maps Z_i to (Z_i, s_i) where $s \overset{\lambda(q)}{\hookrightarrow}_\psi s_i$, for $i = 1, 2, \dots, m$. Recall that λ is the function which labels every control state of $\mathcal{A}$ with a subset of atomic propositions in AP.
  - $([(q, V, q')], s) \overset{\bot}{\hookrightarrow}_3 \chi$ where $\chi : \mathit{Succ}_{CA}([(q, V, q')]) \to (CL \times (S_\psi \cup \{\sharp\}))$ maps each Z_i to $(Z_i, \sharp)$ for $i = 1, 2, \dots, m$.

  Further, we have $([(q, V, q')], \sharp) \overset{\bot}{\hookrightarrow}_3 \chi$ where $\chi : \mathit{Succ}_{CA}([(q, V, q')]) \to (CL \times (S_\psi \cup \{\sharp\}))$ maps each Z_i to $(Z_i, \sharp)$ for $i = 1, 2, \dots, m$.
- Suppose $[(q, V, q')]$ is in CL where (q, V, q') is an infeasible configuration. Then $([(q, V, q')], s) \overset{\bot}{\hookrightarrow}_3 \chi$, where $\chi : \{[(q, V, q')]\} \to (CL \times (S_\psi \cup \{\sharp\}))$ maps each $[(q, V, q')]$ to $([(q, V, q')], \sharp)$.

Intuitively, the tree automaton $\mathcal{R}_3$ simulates $\mathcal{R}_\psi$ along the paths in which every node is labelled $\top$ and assign to each node σ in such a path the corresponding *unique* state reached by $\mathcal{R}_\psi$ upon reading the atomic propositions of v. The states of the form $([(q, V, q')], \sharp)$ in $\mathcal{R}_3$, where $[(q, V, q')] \in CL$, are used to indicate that the node being read is labelled $\bot$ and hence is irrelevant.

Since $\mathcal{R}_{ctrl}$ is the intersection of $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{R}_3$, it is now routine to verify that the language of $\mathcal{R}_{ctrl}$ is nonempty iff there exists a cluster-respecting ψ-winning strategy for $\mathcal{A}$. This establishes theorem 5.4(i), owing to lemma 5.5.

We analyze the complexity of the above decision procedure for theorem 5.4. The non-deterministic Büchi automaton $\mathcal{B}_\psi$ will have $N_{\mathcal{B}_\psi} = 2^{O(|\psi|)}$ states, where $|\psi|$ is the size of ψ. Hence the deterministic Rabin automaton for $\mathcal{R}_\psi$ has $N_{\mathcal{R}_\psi} = 2^{O(N_{\mathcal{B}_\psi} \cdot \log N_{\mathcal{B}_\psi})}$ states and $K_{\mathcal{R}_\psi} = O(N_{\mathcal{B}_\psi})$ accepting pairs ([68]). It is easy to construct the non-deterministic Rabin tree automaton $\mathcal{R}_{ctrl}$ directly which will have $N_{\mathcal{R}_{ctrl}} = O(|CL| \cdot N_{\mathcal{R}_\psi})$ states and $K_{\mathcal{R}_{ctrl}} = O(N_{\mathcal{B}_\psi})$ accepting pairs. We have $|CL| = O(|Q|^2 \cdot (\gamma_{max} \cdot \Gamma^{-1})^n)$. The complexity for testing nonemptiness of $\mathcal{R}_{ctrl}$ ([23]) is $\left(N_{\mathcal{R}_{ctrl}} \cdot K_{\mathcal{R}_{ctrl}}\right)^{O(K_{\mathcal{R}_{ctrl}})}$, that is,

$$\left(O(|Q|^2 \cdot (\gamma_{max} \cdot \Gamma^{-1})^n \cdot 2^{O(|\psi| \cdot 2^{O(|\psi|)})}\right)^{O(2^{|\psi|})} .$$

5.6.3. Synthesis of Controllers.

To prove theorem 5.4(ii), we suppose the set of $\{\top, \bot\}$-labelled $\mathcal{T}$ trees accepted by $\mathcal{R}_{ctrl}$ is nonempty. Then by Rabin's tree theorem [63], the decision procedure for testing the nonemptiness of the language of $\mathcal{R}_{ctrl}$ yields a *regular* $\{\top, \bot\}$-labelled CL-tree $(\mathcal{T}, \eta)$ that is accepted by $\mathcal{R}_{ctrl}$. More precisely, the set of nodes in $\mathcal{T}$ that are labelled by $\top$ by η is a regular subset of $CL^\star$, and moreover $(\mathcal{T}, \eta)$ is presented in the form of a finite transition system, which we shall extend to be an RDA $\mathcal{C}$ with no sensing delay.

Each state of $\mathcal{C}$ will be a node in $\mathcal{T}$ (that is labelled $\top$ by η) and it will be accompanied by a labelling function ξ. The initial location will be ε with $\xi(\varepsilon) = [(q_{in}, V_{in}, q_{in})]$ whereas to all other locations ξ will assign a cluster in CL.

For a state s of $\mathcal{C}$ that is labelled $[(q, V, q')] \in CL$ by ξ, we define its rate function to be ρ_q. Further, we set the invariant I_s to be such that: a valuation $\widehat{V}$ satisfies I_s iff $\left\|\widehat{V}\right\| = \|V\|$. Clearly the invariant I_s can be effectively computed, albeit that I_s is a finite conjunction of inequalities of the form $x_i \prec c$ where c is a rational *or* $c = c' + \ln c''$ with c', c'' are rationals. For each transition of $\mathcal{C}$, we associate it with the guard true. We set the delay parameters and range parameters of $\mathcal{C}$ to be the same as those of $\mathcal{A}$. This completes the proof of theorem 5.4(ii).

5.6.4. QPLTL Specifications.

We note that for a QPLTL formula ψ, one can effectively construct a non-deterministic Büchi automaton $\mathcal{B}_\psi$ over 2^{AP} which accepts precisely the set of models of ψ. It follows that the proof of theorem 5.4 implies:

Theorem 5.6.

(i) *Given a pair $(\mathcal{A}, \psi)$, where $\mathcal{A}$ is an RDA with $\delta^0_{ob} = 1 = \delta^1_{ob}$ and ψ is a QPLTL formula over AP, we can effectively determine whether there exists a ψ-controller for $\mathcal{A}$.*

(ii) *Further, if there exists a ψ-controller for $\mathcal{A}$, then we can effectively construct one in the form of a (finite) RDA with no sensing delay.*

5.7. Discussion

We have considered linear time specifications. One could also study controller synthesis of RDAs with branching time specifications in CTL, or CTL* [21].

It would be interesting to extend our present results to study controller synthesis for the variant of RDAs with finite precision and polynomial guards in [2]. Here, again the controller knows only the measured values of continuous variables and hence does not have "accurate" information about the status of continuous variables. Nevertheless, we believe that the techniques in this chapter would turn out to be useful.

CHAPTER 6

Conclusions

We have explored controlled synthesis in distributed, real-time and hybrid settings.

In the distributed setting, we obtained decidability results on distributed controller synthesis for the large class of CCPs for both robust linear time specifications and branching time specifications given as MSO formulae over event structures. Further, we showed that finite state distributed controllers can be effectively synthesized whenever they exist.

Distributed protocols are often difficult to design by hand. Our results on CCPs imply that one can in fact automatically synthesize many distributed protocols from protypes and specifications. One can also hope that, for specialized kinds of properties such as safety and liveness properties, the complexities of our decision algorithms for distributed controller synthesis can be significantly improved.

In fact, we established the strong result that the MSO theory of every CCP is decidable, where the MSO logic of a CCP is its canonical non-interleaved branching time logic. This result is also of fundamental importance to model checking of CCPs. It serves as the cornerstone for deriving branching time logics that are amenable to more efficient model checking procedures.

It would be interesting to extend the class of CCPs to timed and hybrid settings. That is, to study networks of timed or hybrid automata whose underlying network of transition systems is a CCP.

In the real-time setting, we showed that admission controllers for real-time systems with tasks is decidable. Further, if there exists an admission controller, then we can effectively synthesize one as a (finite) timed automaton. Clearly we can apply these results to synthesize admission controllers if a given task arrival pattern is not schedulable with respect to a fixed processor and the preemptive EDF scheduling policy.

We have assumed that the task arrival pattern is described by a single timed automaton. It would be interesting to consider to a task arrival pattern to be given by a network of timed automata and to synthesize a family of admission controllers, one for each component timed automaton.

We have assumed that the scheduling policy is fixed. A more ambitious problem is to synthesize a scheduling policy for a given task arrival pattern on fixed computing resources.

We have assumed that there is a processor dedicated to all the tasks. If the processor is not dedicated, one may ask the question whether there exists an admission controller that guarantees schedulability of accepted task instances and that the QoS specification is met, but tries to minimize the computing load on the processor.

In the hybrid setting, we showed the control state sequence languages of an RDA is regular and solved the controller synthesis problem for RDAs with no sensing delays.

The regularity result can be applied to solve model checking problems for RDAs. The techniques for proving this regularity result also provide insights for handling variables evolving at differential rates. It would be interesting to study whether RDAs can be used to abstract hybrid automata where the rates of variables are governed by linear differential equations.

It would be worthwhile to try applying the controller synthesis result for RDAs to in the design of practical control systems. For safety, liveness and other simple specifications, it is promising that we can improve the complexities of our decision procedure for controller synthesis of RDAs with no sensing delays.

We have focused on decidability and undecidability results of controller synthesis in distributed, real-time and hybrid settings. As with most decision procedures for controller synthesis, our decision algorithms are of rather high complexity theoretically and thus not immediately feasible for practical applications. The search of pragmatically feasible algorithms is a common challenge in the field of controller synthesis. However, one should not be pessimistic, since the average case complexities of these algorithms could be far less than their worst case complexities.

Bibliography

[1] Yasmina Abdeddaïm, Eugene Asarin, and Oded Maler. Scheduling with timed automata. *Theoretical Comp. Sci.*, 354(2):272–300, 2006.

[2] M. Agrawal, F. Stephan, P.S. Thiagarajan, and S. Yang. Behavioural approximations for restricted linear differential hybrid automata. In *HSCC '06, LNCS 3927*, pages 4–18. Springer, 2006.

[3] M. Agrawal and P.S. Thiagarajan. Lazy rectangular hybrid automata. In *7th HSCC, LNCS 2993*, pages 1–15. Springer, 2003.

[4] M. Agrawal and P.S. Thiagarajan. The discrete time behaviour of lazy linear hybrid automata. In *8th HSCC, LNCS 3414*, pages 55–69. Springer, 2005.

[5] K. Altisen, G. Gössler, and J. Sifakis. Scheduler modeling based on the controller synthesis paradigm. *Real-Time Systems*, 23:55–84, 2002.

[6] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, Sifakis J., and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Comp. Sci.*, 138:3–34, 1995.

[7] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Comp. Sci.*, 126:183–235, 1994.

[8] E. Asarin, O. Bournez, T. Dang, O. Maler, and A. Pnueli. Effective synthesis of switching controllers for linear systems. *Proc. of IEEE*, 88:1011–1025, 2000.

[9] E. Asarin, O. Maler, and A. Pnueli. Symbolic controller synthesis for discrete and timed systems. In *Hybrid Systems II, LNCS 999*, pages 1–20. Springer, 1995.

[10] H. Ben-Abdallah, J. Choi, D. Clarke, Y. Kim, I. Lee, and H. Xie. A process algebraic approach to the schedulability analysis of real-time systems. *Real-Time Systems*, 15:189–219, 1998.

[11] J. Bengtsson and W. Yi. On clock difference constraints and termination in reachability analysis of timed automata. In *ICFEM '03, LNCS 2885*, pages 491–503. Springer, 2003.

[12] V. Bertin, M. Poize, and J. Sifakis. Towards validated real-time software. In *12th Euromicro Conf. on Real Time Sys.*, pages 157–164, 2000.

[13] P. Bouyer, D. D'Souza, P. Madhusudan, and A. Petit. Timed control with partial observability. In *15th CAV, LNCS 2725*, pages 180–192. Springer, 2003.

[14] J.R. Büchi and L.H. Landweber. Solving sequential conditions by finite-state strategies. *Trans. of AMS*, 138:295–311, 1969.

[15] G.C. Buttazzo. *Hard real-time computing systems : Predictable scheduling algorithms and applications.* Kluwer, 1997.

[16] A. Church. Logic, arithmetic, and automata. In *Int. Cong. Math. 1962*, pages 21–35, 1963.

[17] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, 1999.

[18] V. Diekert and G. Rozenberg, editors. *The Book of Traces.* World Scientific, Singapore, 1995.

[19] D. D'Souza and P. Madhusudan. Timed control synthesis for external specifications. In *STACS '02, LNCS 2285*, pages 571–582. Springer, 2002.

[20] A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *Int. J. of Game Theory*, 8:109–113, 1979.

[21] E.A. Emerson. Temporal and modal logics. In *Handbook of Theoretical Comp. Sci., Vol. B*, pages 997–1072. Elsevier, 1990.

[22] E.A. Emerson and E.M. Clarke. Using branching time logic to synthesize synchronization skeletons. *Science of Comp. Prog.*, 2:241–266, 1982.

[23] E.A. Emerson and C.S. Julta. The complexity of tree automata and logics of programs. In *29th FOCS*, pages 328–337. IEEE Press, 1988.

[24] E. Fersman, P. Pettersson, and W. Yi. Timed automata with asynchrounous processes: Schedulability and decidability. In *8th TACAS, LNCS 2280*, pages 67–82. Springer, 2002.

[25] P. Gastin, B. Lerman, and M. Zeitoun. Distributed games and distributed control for asynchronous systems. In *LATIN '04, LNCS 2976*, pages 455–465. Springer, 2004.

[26] P. Gastin, B. Lerman, and M. Zeitoun. Distributed games with causal memory are decidable for series-parallel systems. In *FSTTCS '04, LNCS 3328*, pages 275–286. Springer, 2004.

[27] T.A. Henzinger. The theory of hybrid automata. In *11th LICS*, pages 278–292. IEEE Press, 1996.

[28] T.A. Henzinger, B. Horowitz, and C.M. Kirsch. Embedded control systems development with GIOTTO. In *LCTES 2001*, pages 64–72. ACM Press, 2001.

[29] T.A. Henzinger, B. Horowitz, and R. Majumdar. Rectangular hybrid games. In *10th CONCUR,LNCS 1664*, pages 320–335. Springer, 1999.

[30] T.A. Henzinger and P.W. Kopke. State equivalences for rectangular hybrid automata. In *7th CONCUR, LNCS 1119*, pages 530–545. Springer, 1996.

[31] T.A. Henzinger and P.W. Kopke. Discrete-time control for rectangular hybrid automata. *Theoretical Comp. Sci.*, 221:369–392, 1999.

[32] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *J. of Comp. and Sys. Sci.*, 57:94–124, 1998.

[33] M. Hirsch and S. Smale. *Differential Equations, Dynamical Systems and Linear Algebra*. Academic Press, 1974.

[34] T.J. Jech. About the axiom of choice. In *Handbook of Math. Logic*, pages 345–370. North-Holland Publishing, 1977.

[35] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine. Integration graphs: A class of decidable hybrid systems. In *Hybrid Systems, LNCS 736*, pages 179–208. Springer, 1993.

[36] T.J. Koo, G.J. Pappas, and S. Sastry. Mode switching synthesis for reachability specifications. In *HSCC '01, LNCS 2034*, pages 333–346, 2001.

[37] K. Kunen. Combinatorics. In *Handbook of Math. Logic*, pages 371–401. North-Holland Publishing, 1977.

[38] O. Kupferman, P. Madhusudan, P.S. Thiagarajan, and M.Y. Vardi. Open systems in reactive environments: Control and synthesis. In *11th CONCUR, LNCS 1877*, pages 92–107. Springer, 2000.

[39] O. Kupferman and M. Vardi. Synthesizing distributed systems. In *LICS '01*, pages 16–19. IEEE Press, 2001.

[40] O. Kupferman and M.Y. Vardi. Synthesis with incomplete information. In *2nd Int. Conf. on Temporal Logic*, pages 91–106, 1997.

[41] H.-H. Kwak, I. Lee, A. Philippou, J.-Y. Choi, and O. Sokolsky. Symbolic schedulability analysis of real-time systems. In *RTSS '98*, pages 409–418. IEEE Press, 1998.

[42] G. Lafferriere, G.J. Pappas, and S. Sastry. O-minimal hybrid systems. *Math. Control Signals Systems*, 13:1–21, 2000.

[43] G. Lafferriere, G.J. Pappas, and S. Yovine. A new class of decidable hybrid systems. In *HSCC '99, LNCS 1569*, pages 137–151. Springer, 1999.

[44] G. Lafferriere, G.J. Pappas, and S. Yovine. Symbolic reachability computation for families of linear vector fields. *J. Symbolic Computation*, 32:231–253, 2001.

[45] J.W.S. Liu. *Real-Time Systems*. Prentice-Hall, 2000.

[46] K. Lodaya, R. Parikh, R. Ramanujam, and P.S. Thiagarajan. A logical study of distributed transition systems. *Information and Computation*, 119:91–118, 1995.

[47] P. Madhusudan. Model-checking trace event structures. In *LICS '03*, pages 371–380. IEEE Press, 2003.

[48] P. Madhusudan and P.S. Thiagarajan. Controllers for discrete event systems via morphisms. In *CONCUR '98, LNCS 1466*, pages 18–33. Springer, 1998.

[49] P. Madhusudan and P.S. Thiagarajan. Distributed control and synthesis for local specifications. In *ICALP '01, LNCS 2076*, pages 396–407. Springer, 2001.

[50] P. Madhusudan and P.S. Thiagarajan. Branching time controllers for discrete event systems. *Theoretical Comp. Sci.*, 274:117–149, 2002.

[51] P. Madhusudan and P.S. Thiagarajan. A decidable class of asynchronous distributed controllers. In *CONCUR '02, LNCS 2421*, pages 145–160. Springer, 2002.

[52] P. Madhusudan, P.S. Thiagarajan, and S. Yang. The MSO theory of connectedly communicating processes. In *FSTTCS '05, LNCS 3821*, pages 201–212. Springer, 2005.

[53] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *STACS '95, LNCS 900*, pages 229–242. Springer, 1995.

[54] Z. Manna and R. Waldinger. A deductive approach to program synthesis. *ACM Trans. on Prog. Lang. and Sys.*, 2:90–121, 1980.

[55] Z. Manna and P. Wolper. Synthesis of communicating processes from temporal logic specifications. *ACM Trans. on Prog. Lang. and Sys.*, 6:68–93, 1984.

[56] R. McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65:149–184, 1993.

[57] T. Moor and J.M. Davoren. Robust controller synthesis for hybrid systems using modal logic. In *HSCC '01,LNCS 2034*, pages 433–446. Springer, 2001.

[58] P. Niebert and S. Yovine. Computing optimal operation schemes for chemical plants in multi-batch mode. In *Hybrid Sys., Comp. and Control, LNCS 1790*, pages 338–351. Springer, 2000.

[59] W. Penczek. Model-checking for a subclass of event structures. In *TACAS '97, LNCS 1217*, pages 146–164. Springer, 1997.

[60] G.L. Peterson and J.H. Reif. Multiple-person alternation. In *FOCS '79*, pages 348–363. IEEE Press, 1979.

[61] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *16th POPL*, pages 179–190, 1989.

[62] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *FOCS '90*, pages 746–757. IEEE Press, 1990.

[63] M. Rabin. Decidability of second order theories and automata on infinite trees. *Trans. of AMS*, 141:1–35, 1969.

[64] M. Rabin. *Automata on Infinite Objects and Church's Problem.* AMS, 1972.

[65] P.J. Ramadge and W.M. Wonham. On the supremal controllable sublanguage of a given language. *SIAM J. Control and Optimization*, 25:206–230, 1987.

[66] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete-event processes. *SIAM J. Control and Optimization*, 25:206–230, 1987.

[67] P.J. Ramadge and W.M. Wonham. The control of discrete event systems. *Proc. of IEEE*, 77:81–98, 1989.

[68] S. Safra. On the complexity of ω-automata. In *29th FOCS*, pages 319–327. IEEE Press, 1988.

[69] V. Sassone, M. Nielsen, and G. Winskel. Models for concurrency: Towards a classification. *Theoretical Comp. Sci.*, 170:297–348, 1996.

[70] O. Shakernia, S. Sastry, and G.J. Pappas. Decidable controller synthesis for classes of linear systems. In *HSCC '00, LNCS 1790*, pages 407–420. Springer, 2000.

[71] A. Stefanescu, J. Esparza, and A. Muscholl. Synthesis of distributed algorithms using asynchronous automata. In *CONCUR 2003, LNCS 2761*, pages 27–41, 2003.

[72] P.S. Thiagarajan. Regular trace event structures. Technical report, 1996. Technical Report RS-96-32, BRICS Tech Report, Denmark.

[73] W. Thomas. Automata on infinite objects. In *Handbook of Theoretical Comp. Sci., Vol. B*, pages 133–192. Elsevier, 1990.

[74] W. Thomas. On the synthesis of strategies in infinite games. In *12th STACS, LNCS 900*, pages 1–13. Springer, 1995.

[75] C. Tomlin, J. Lygeros, and S. Sastry. Computing controllers for nonlinear hybrid systems. In *HSCC '99, LNCS 1569*, pages 238–255. Springer, 1999.

[76] C. Tomlin, J. Lygeros, and S. Sastry. A game theoretic approach to controller design for hybrid systems. *Proc. of IEEE*, 88:949–970, 2000.

[77] S. Tripakis. Decentralized control of discrete event systems with bounded or unbounded delay communication. *IEEE Trans. on Automatic Control*, 49:1489–1501, 2004.

[78] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115:1–37, 1994.

[79] I. Walukiewicz and S. Mohalik. Distributed games. In *FSTTCS '03, LNCS 2914*, pages 338–351. Springer, 2003.

[80] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56:72–79, 1983.

[81] W. Zielonka. Notes on finite asynchronous automata. *R.A.I.R.O.—Inform. Théor. Appl.*, 21:99–135, 1987.

[82] U. Zwick and M.S. Paterson. The complexity of mean payoff games on graphs. *Theoretical Comp. Sci.*, 158:343–359, 1996.