Modeling and qualitative simulation of hybrid systems

Hadi Zaatiti

université PARIS-SACLAY

NNT: 2018SACLS493

Modélisation et simulation qualitative de systèmes hybrides

Thèse de doctorat de l'Université Paris-Saclay préparée à:

Université Paris-Sud

Commissariat à l'Énergie Atomique

École doctorale n° 580
Sciences et technologies de l'information et de la communication (STIC)

Spécialité de doctorat: Informatique

Thèse présentée et soutenue à Orsay, le 29 novembre 2018, par:

Hadi Zaatiti

Composition du jury:

Sylvain Conchon Professeur, Université Paris-Sud	Président
Walid Taha Professeur, Université de Halmstad	Rapporteur
Goran Frehse Professeur, ENSTA-ParisTech	Rapporteur
Erika Ábrahám Professeur, Université RWTH Aachen	Examinatrice
Philippe Dague Professeur, Université Paris-Sud	Directeur
Jean-Pierre Gallois Ingénieur, Commissariat à l'Énergie Atomique	Co-encadrant

Thèse de doctorat

Acknowledgements

I am forever in debt to my professor Philippe Dague for guiding me throughout the thesis. He provided me with constant feedback, was always available for any inquiry and showed me new horizons to extend the research. Without his help the current quality of the thesis work and the present manuscript would have never been achieved.

I thank my supervisor Jean-Pierre Gallois for the daily discussions about the thesis topic and his constant supervision over the works. My sincerest thanks goes also to Lina Ye for bringing her expertise in the diagnosability verification domain, her work in co-authoring the publications and her numerous advice.

The help I received at CEA from my colleagues Arnault and Stéphane during my implementation work was of major importance, for that I thank them indefinitely.

I address my profound thanks and gratitude to professor Walid Taha and professor Goran Frehse for taking the time to evaluate my manuscript, their valuable comments and for taking part of the jury for my thesis defence. I also thank professor Erika Ábrahám and professor Sylvain Conchon for accepting to take part in the jury and assisting to the oral defense.

I thank my family, my mother Hala, father Samir and sister Saly living in my home country Lebanon for their eternal support and their visits to France for checking up on me constantly.

My thanks goes also to my friends Ben, Jad, Houssam, Nadine, Ghida, Dory, Slim, Lamia, Minh-Thang and Imene for being there for me in time of need.

Abstract

Hybrid systems are complex systems that combine both discrete and continuous behaviors. Verifying behavioral or safety properties of such systems, either at design stage or on-line is a challenging task. Actually, computing the reachable set of states of a hybrid system is undecidable. One way to verify those properties over such systems is by computing discrete abstractions and inferring them from the abstract system back to the original system. We are concerned with abstractions oriented towards hybrid systems diagnosability checking. (Bounded)-diagnosability can be see as the ability of the system (or an extended system) to localize a problem in bounded time such as the occurrence of a fault. Our goal is to create discrete abstractions in order to verify if a fault that would occur at runtime could be unambiguously detected in finite time by the diagnoser. This verification can be done on the abstraction by classical methods developed for discrete event systems, which provide a counterexample in case of nondiagnosability. The absence of such a counterexample proves the diagnosability of the original hybrid system. In the presence of a counterexample, the first step is to check if it is not a spurious effect of the abstraction and actually exists for the hybrid system, witnessing thus non-diagnosability. Otherwise, we show how to refine the abstraction and continue the process of looking for another counterexample.

Contents

Chapter 1

Introduction

The thesis is concerned with the formal verification of complex systems at modeling stage. Precisely, the complex systems dealt with are hybrid systems that exhibit a twofold behavior: continuous and discrete. Hybrid Automata are a common framework to model hybrid systems using finitely many discrete modes and continuous real-valued variables. They provide an intuitive, powerful and general way to model a large number of practical systems, applicable in many domains: aeronautics, railway, biology and more generally cyber-physical systems (CPS) where control loops act upon digital electronic circuits (discrete behavior) and are orchestrated by the physical environment (continuous processes).

The problem of hybrid systems verification is addressed, i.e., given a hybrid automaton model and a property, does the model verify this property? The literature shows that the verification of hybrid systems is challenging. In fact, the study of hybrid systems requires cross-disciplinary competences. Consequently, verification methods must couple and extend results originating from different domains such as control theory, applied mathematics and both theoretical and practical computer science. Moreover, due to the expressiveness of the hybrid automata modeling language, verifying the most simple properties (such as reachability) is undecidable for hybrid systems. Progress in hybrid systems is also held back by the lack of a strong standardized specification and modeling language. Many tools for hybrid systems verification exist today, each of them having its own input language and addressing a specific problem with particular methods.

The work addressed by this thesis is the application of principles from the

qualitative reasoning domain to hybrid systems verification. Intuitively, qualitative reasoning aggregates similar behaviors of the system into representative sets and reasons over these sets. The relations describing the representative sets allow a global overview of the system behavior, performing as such, what is called in the literature, a qualitative simulation. For hybrid systems, it is difficult to aggregate any exact set of behaviors. For this reason, representative sets that are over-approximations of the real system behaviors are used. The overview mapping is thus an abstraction relating the real (or concrete) system and the related representative (or abstract) sets. The abstract system being an over-approximation, a class of properties can be verified using the abstraction and inferring the result to the concrete system.

The topic of hybrid systems verification being too general, a particular property is addressed: diagnosability. This property of the system supposes that a given hybrid automaton models, not only the correct behavior of the system, but faulty ones as well. Practically speaking, if the system is inevitably unable to avoid the occurrence of a fault(s) and if the faulty behavior can be characterized in advance, one can model this faulty behavior. The system is considered diagnosable if it is able to detect the faulty behavior and identify the fault that occurred among the set of possible faults after the fault occurrence. Moreover, the system is supposed partially observable, thus, diagnosability analysis should only use the partial observations to establish its verdict. Our goal is to create discrete abstractions in order to verify, at design stage, if a fault that would occur at run time could be unambiguously detected in finite time (or within a given finite time bound for bounded diagnosability) by the diagnoser using only the allowed observations. This verification can be done on the abstraction by classical methods developed for discrete event systems, which provides a counterexample in case of non-diagnosability. The absence of such a counterexample proves the diagnosability of the original hybrid system. In presence of a counterexample, the first step is to check if it is not a spurious effect of the abstraction and actually exists for the hybrid system, witnessing thus non-diagnosability. Otherwise, the abstraction is to be refined, guided by the elimination of the counterexample, and the process of looking for another counterexample continues until some precision is reached.

1.1 Contributions

The contributions of the thesis are:

- The elaboration of a method that performs a qualitative simulation of a given hybrid system. The method incorporates reasoning from the qualitative domain and extend previous results from the literature.
- The implementation of a tool that performs the qualitative simulation of the hybrid system given a polynomial hybrid automaton as input.
- An extension to the qualitative simulation framework for producing timed abstractions of a given hybrid system using existing flow-pipe computation techniques. The flow-pipe computation is guided by the resulting qualitative simulation. Timed automata are used to represent the qualitative timed abstraction.
- The elaboration of a method that applies the results issued from the qualitative simulation and timed abstractions to diagnosability verification of hybrid systems in the framework of counter-example guided abstraction refinement (CEGAR) loop. Methods that were specific to diagnosability analysis of discrete and timed automata are used.

1.2 Thesis outline

The upcoming chapters are organized as follows:

- **Chapter 2** presents the scientific context. The literature concerned with model driven engineering, qualitative reasoning and simulation, hybrid system verification techniques for reachability and diagnosability verification and the existing tools and challenges are reviewed.
- **Chapter 3** introduces a formal framework for hybrid system verification, fault modeling and diagnosability. The adopted notations are introduced and practical examples from known benchmarks are used to illustrate the definitions.

- **Chapter 4** presents qualitative reasoning based abstraction methods for hybrid systems and their applicability to diagnosability verification. The method combines predicate abstraction with qualitative based reasoning and is instantiated to the class of polynomial hybrid automata.
- **Chapter 5** presents the tool for performing a qualitative simulation of a given hybrid system illustrated and tested on different examples.
- **Chapter 6** concludes the thesis. The chapter reviews the contributions while comparing with closely related work from other authors and finally draws some perspectives.

Résumé français

1.3 Summary in french

Cette partie résume en français le contenu de la thèse. On présentera d'abord la structure du manuscrit et le contenu des chapitres puis le résumé des publications.

1.3.1 Mot-clés

Systèmes et automates hybrides, raisonnement qualitatif, diagnosticabilité, systèmes à évènements discrets, automate temporisé, techniques d'abstraction.

1.3.2 Structure de la thèse

Le manuscrit de thèse est constitué de six chapitres et deux annexes.

- **Chapitre 1 :** Le premier chapitre est introductif. Il décrit la problématique traitée par la thèse et présente le contexte scientifique général ainsi qu'une liste des contributions de la thèse.
- **Chapitre 2 :** Le deuxième chapitre fournit un aperçu global des différentes problématiques évoquées par la thèse et les éléments de l'état de l'art sur lesquels se fondent les travaux de la thèse : la modélisation et l'analyse des systèmes hybrides, la modélisation et la simulation qualitatives et la diagnosticabilité. Cette partie résume les concepts de base pour permettre au lecteur de se familiariser avec la thématique. Les concepts sont illustrés par des exemples.

- **Chapitre 3 :** Le troisième chapitre présente le formalisme de base utilisé pour la description des systèmes hybrides : les automates hybrides. Le problème de diagnosticabilité sous observation partielle, formalisé en termes d'automates hybrides, est également présenté, suivi des classes particulières de systèmes hybrides notamment les automates temporisés et les automates hybrides polynomiaux qui nous intéressent pour les travaux de la thèse.
- **Chapitre 4 :** Le quatrième chapitre présente une approche pour construire des abstractions temporisées d'un automate hybride et son application à la diagnosticabilité. Cette abstraction peut être raffinée, à la précision souhaitée, en partitionnant davantage l'espace d'états. Ensuite cette approche est utilisée pour résoudre le problème de diagnosticabilité : deux copies de l'automate hybride sont utilisées pour former un nouveau système de jumeaux (twin plant). Ainsi on peut caractériser un problème de diagnosticabilité comme une exécution du système dans laquelle une copie atteint un état fautif et l'autre non, les deux produisant les mêmes observations sur l'horizon de temps donné. L'abstraction est utilisée pour permettre l'analyse des automates hybrides avec une dynamique polynomiale.
- **Chapitre 5 :** Ce chapitre présente les expériences sur des études de cas. Un outil prototype qui permet d'effectuer la simulation qualitative d'une manière automatique est présentée. L'architecture de l'outil est présentée puis une étude des performances avec les temps de calculs évalués sur plusieurs modèles de systèmes hybrides et continus est établie.
- **Chapitre 6 :** Le chapitre final présente les conclusions. Une comparaison des travaux avec d'autres travaux proches de la thématique considérée est présentée. Finalement, des perspectives envisageables sont illustrées.

1.3.3 Résumé des publications

• [109] Les systèmes hybrides exhibent une interaction entre des décisions de contrôle discret et des processus physiques continus, ces systèmes sont au coeur des systèmes dits cyber-physiques. Suite à ces interactions, la vérification de ces systèmes est difficile. Dans cet article, on s'intéresse aux techniques d'abstraction des systèmes hybrides. Les abstractions sont utiles pour automatiser le processus de vérification du système hybride, comme la vérification de la sûreté ou même des propriétés plus compliquées lorsque la vérification est combinée avec des algorithmes de "model checking" classiques. L'article présente un outil qui calcule de façon automatique l'abstraction d'un système hybride donné en entrée, celui-ci ayant une expressivité polynomiale. L'abstraction peut être manuellement raffinée afin d'atteindre une meilleure précision, cette opération est guidée par le concepteur du système. L'outil est testé sur plusieurs exemples.

• [110] La vérification des propriétés des systèmes hybrides depuis leur conception, comme la sûreté ou la diagnosticabilité, ou pendant leur exécution, telle que la détection et l'isolation de fautes, est une tâche difficile. Dans ce papier on est concerné par les techniques d'abstraction orientées pour la vérification de la diagnosticabilité d'un système hybride donné. La vérification est effectuée sur l'abstraction utilisant des méthodes classiques développées pour les systèmes à évènements discrets étendues avec des contraintes temporelles, qui fournissent un contre-exemple dans le cas où le système n'est pas diagnosticable. L'absence d'un tel contre-exemple prouve la diagnosticabilité du système original. En présence d'un contre-exemple, la première étape est de vérifier si celui-ci n'est pas un faux contre-exemple résultant de l'abstraction et qu'il figure bien au niveau concret, témoignant ainsi de la non-diagnosticabilité. Dans le cas contraire, on montre comment raffiner l'abstraction guidé par l'élimination du contre-exemple puis le processus de recherche d'un nouveau contre-exemple continue jusqu'à

atteindre un résultat ou un verdict non concluant. On se sert des principes de modélisation et raisonnement qualitatifs pour le calcul de l'abstraction discrète. Les abstractions en tant qu'automate temporisé sont particulièrement étudiées, en effet elles permettent de gérer les contraintes temporelles qui sont capturées à un niveau qualitatif du système hybride.

• [111] La croissance en complexité des systèmes rend plus difficile la détection et l'isolation de fautes. Ce fait s'applique aux systèmes hybrides qui combinent un double aspect discret et continu. La vérification de propriétés durant la phase de conception du système, telles que la sûreté, la diagnosticabilité et la prédictabilité, ou durant leur exécution, telles que la détection et l'isolation de fautes, est une tâche difficile. En effet, calculer l'ensemble des états atteignables d'un système hybride donné n'est pas décidable, ceci est principalement dû à la grande expressivité du langage de modélisation des systèmes hybrides notamment en ce qui concerne la partie continue. Une méthode employée pour vérifier les propriétés de tels systèmes consiste à calculer une abstraction discrète du système considéré, à appliquer le processus de vérification à l'abstraction et à inférer le résultat obtenu au système hybride original. La diagnosticabilité est une propriété qui décrit la capacité d'un système à déterminer l'occurrence d'une faute à partir des observations. Ce problème a reçu une attention considérable dans la littérature. Néanmoins, la plupart des travaux existants sont appliqués aux systèmes à évènements discrets ou, dans une moindre mesure, aux systèmes continus mais très peu aux systèmes hybrides. Dans ce chapitre d'ouvrage, on est concerné par les méthodes d'abstraction orientées pour la vérification de la diagnosticabilité d'un système hybride donné. Notre objectif est de créer des abstractions discrètes pour vérifier, dès la modélisation de ce système, si une faute qui peut avoir lieu durant l'exécution du système peut-être détectée sans ambiguïté en temps fini (temps borné donné) par le diagnostiqueur en utilisant seulement les observations disponibles. La vérification peut être effectuée

sur l'abstraction en utilisant des méthodes classiques développées pour les systèmes à évènements discrets, qui fournissent un contre-exemple si le système n'est pas diagnosticable. En présence d'un contre-exemple, la première étape est de vérifier s'il ne résulte pas de l'abstraction et existe au niveau du système hybride témoignant ainsi de la non-diagnosticabilité du système. Dans le cas contraire, on montre comment raffiner l'abstraction, guidé par l'élimination du contre-exemple et on poursuit le processus de recherche d'un autre contre-exemple jusqu'à l'obtention d'un résultat final ou d'un verdict non concluant. On se sert des principes du raisonnement et modélisation qualitatifs pour calculer l'abstraction discrète et on définit plusieurs stratégies de raffinement. Les abstractions en tant qu'automate temporisé sont particulièrement étudiées, vu qu'elles permettent de représenter qualitativement les contraintes temporelles.

Chapter 2

Scientific Context

In this chapter the scientific context related to the thesis work is introduced. Section 2.1 presents model driven engineering and verification. Section 2.2 reviews the literature concerned with qualitative reasoning and simulation. Section 2.3 addresses hybrid systems modeling and verification techniques. The last section summarizes existing tools and challenges in the hybrid systems verification domain.
The objective of the thesis is the study of abstraction techniques of hybrid systems using qualitative principles and their application to diagnosability verification. For this reason, it is important to review the concerned literature and existing techniques presented in this chapter. Consequently the literature addressing qualitative reasoning principles is presented. The latter are used in the proposed abstraction technique. Moreover, existing hybrid systems verification techniques are reviewed to bring forward what the elaborated qualitative abstraction can complement and for comparison purposes. Works addressing diagnosability of discrete event, timed and hybrid systems are recalled; in fact the proposed diagnosability verification technique uses and extends some of the thesis works from the presented scientific context.

2.1 Modeling and verification

Models today are present in every scientific discipline to reason, exchange information and withdraw conclusions from any process. The act of modeling involves

three stakeholders: what is being modeled or the system to be modeled S, the model M and the modeling language (or paradigm). Given S and the chosen modeling language(s), the act of modeling produces M. Practically speaking, M holds information about S which often is not all the information contained within S. In other words, a model is never complete unless it is the actual system [76]. This lack of information between the model and the system translates naturally to a distance indicating how faithful the model is towards S for a given purpose. A portrait photo can be seen as a model of a person, illustrating the facial form, skin tone, eye color and so on, it is a faithful model if the purpose is to illustrate the physical characteristics of the person. However it is not the case for representing personality traits of the person such as likes or dislikes and interests.

Usage In scientific disciplines, models serve various purposes such as communication and collaboration between teams, testing and prototyping, simulation and verification. The use of models today is central for any scientific and research activity. Engineers use models as specifications to build systems in the future that do not exist today, chemists use models to describe observations from real experiments, physicists use models to predict future behaviors or trace back the unknown history of a physical process of the environment [76]. Today, modeling languages are numerous and can range from application specific languages to domain specific ones.

Deterministic models If a model can react to specific actions, then a deterministic model is one that reacts the same way for the same action. For example, a function that always gives the same output for a given input is a deterministic function. A function that could provide one of two possible outputs for a given input is a non-deterministic function. Determinism is useful for modeling systems that necessarily exhibit some behavior. Non determinism is also useful in representing systems with uncertainties, where the real system may exhibit many "correct" behaviors.

Abstracting and modeling A model can be seen as an abstract representation of S. The model can carry a wide variety of information, one can further abstract

the model to see a specific aspect of the system such as abstracting the model of a building to see the places of wires. Such abstraction of the model is useful for the electrical engineer. In the computer science field and the early design phase of a large software, it comes as a first step to build a conceptual model of the software. Such a model does not represent the underlying code or programming language and libraries used, but only the broad idea and intent for which the software serves and clarifying all idea-related ambiguities. It thus specifies what is the software supposed to do and not how and is certainly an abstraction of it. Different modeling languages such as the Unified Modeling Language (UML) are used for conceptual modeling.

2.1.1 Model checking

In engineering fields, modeling is often conceptual at the early design stage. The real system is then built while being faithful to the model. One would like to test or verify that the conceptual model behaves according to a given specification [34].

Verification and Validation Given a model M and a verification purpose V (a specification) then model checking refers to deciding whether or not V is verified by M denoted $M \models V$. This process is called model checking. It refers to methods and algorithms for exploring M and determining if it obeys the specification of its intended behavior. Research in model checking aims to automate the verification process of M. Note that model checking does not take into account mistakes that can occur at the modeling phase in which case the model is not credible with respect to the actual system S. The process of checking if the system designer built the right model is referred to as system validation. Model checking attracted interest in industry, providing, at design stage, error-prone software and products. However model checking has been held back by the state explosion problem, which is the problem that the number of states in a system grows exponentially in the number of system components. Much research has been devoted to ameliorating this problem via abstraction techniques and compositional reasoning [50].

Symbolic Model Checking Around 1990, techniques that used symbolic state space exploration arose. If the system is modeled as a simple automaton with states and transitions, then the verification procedure is carried out by reasoning over a set of states [82]. BDDs (binary decision diagrams) encode these sets of states using functions and allow one to compute transitions based on them rather than on individual states [1]. These techniques allowed one to handle larger designs and target a greater class of complexity through the use of abstraction and compositional reasoning. Nonetheless, they lack applicability due to each model application requiring a reasoning on its own, as to how to group the states and finding equivalence classes. Today many arising projects observe collaborations between large industrial groups working in the same field. The aim is to obtain application specific verification tools in avionics, autonomous vehicles, railway and other system manufacturing domains where the human knowledge and expertise are directly taken into account in the elaborated tools.

Bounded Model Checking (BMC) [33] Bounded model checking refers to verifying a property up to a tolerated bound. In a simple automaton, the bound can be expressed by the maximum number of transition that can be taken starting from some initial state. BMC was applied to reachability and liveness properties where in the first, all generated sequences of states are within a given set of states and the second is verified by detecting whether or not we can find a loop in the sequences. Many results have been obtained in safety verification, where invariants (detailed in the next paragraph) play a major role. An invariant is expressed as a property that holds in each and every state of the automaton. Compared to BDD, BMC is efficient when searching for a counterexample of the property to verify, which is expected since we are adding a bound when exploring the state space. Some of the disadvantages of bounded model checking is that the method lacks completeness and the types of properties that can currently be checked are very limited. In bounded model checking only finite length sequences are explored, so only those safety properties for which it is enough to look at only bounded length sequences may be entirely verified. For the thesis work, a particular case of the diagnosability property can be seen as a bounded model checking problem. It

is the case when verifying bounded diagnosability, i.e. the system's capacity of detecting a fault in a bounded time after its occurrence.

2.1.2 Invariants

Invariants play a key role in verification methods. We show here the different facets and characteristics of invariants.

Invariant Let us apply repeatedly to a real input x, a function $f(x) = 3x - 1$. If the initial value is $x = 1$ then an invariant is an assertion which holds indefinitely in a certain context, e.g., $\phi = x > 0$ is considered as an invariant for the system. It is called invariant because it is not changing with respect to time. In this example, time is implicit and can be measured discretely by the number of times we applied f to the input x. The link with verification methods is the possibility of directly using found invariants of the model against the given specification. In general models, finding invariants is an extremely challenging task. Today there is no general tools that synthesize model invariants, even for loop invariants in programs. Most of the existing program verification tools require human intervention for manually inserting invariants to deduce the needed verification proof. The difficulty arises in the fact that modeling languages with large expresiveness are needed. Consequently designing verification methods that can handle all the possible situations induced by the expressive language becomes complex.

Inductive Invariant Let us apply f for k times where $k \in \mathbb{N}_+$. Does knowing that ϕ is verified on the $k-1th$ iteration enough to decide if ϕ is true on the kth iteration? For $\phi = x > 0$ this cannot be true, take $x = 0.2$ as initial value. An inductive invariant, is an invariant that requires no further assumptions or other conditions to be met for it to be indefinitely true. Given this definition, we can consider $\phi_{ind} = x > 1$ as inductive invariant.

Fixed Points Given a function $f : X \to X$, an element $x \in X$ is a fixed point of f if $f(x) = x$. Hence x is invariant by f. Many problems in mathematics and computer science can be formulated in terms of the existence of a fixed point.

Application to computer science Due to the increasing complexity of computer programs, rigorous methods for error detection are needed. Invariants and fixed points are important in error detection in computer programs. Theorem proving over computer programs requires finding complex invariants, which is a very difficult task that, most of the time, cannot be automated by a program itself. In computer science, semantics is the mathematical study of the meaning of programming languages. They describe the processes a computer follows when executing a program in a specific language. Verification tasks over computer programs often involves abstracting the semantics of the program. This field is known as the Abstract Interpretation, the theory for approximating semantics of programming languages. This theory allows us to rigorously describe the idea that a certain semantic can be more or less precise depending on the level of observation. Many reasoning present in program verification have been applied to hybrid systems verification.

2.1.3 Satisfiability Solvers

A solver is a mathematical software that takes as input a mathematical representation of a problem and tells whether or not the problem admits a solution. In any verification problem, a solver is usually used to find solutions given a set of constraints. Some of these problems is solving equations, applying sorting or shortest path algorithms, finding the maximum and minimum values of a table or a continuous multi-variable function and so on. *SAT* or Satisfiability Problem can be summarized as: Given a formula ϕ with specified variables, the goal is to determine whether exists or not an interpretation for all variables for which ϕ is satisfied. Example of a sat problem: Find $x \in \mathbb{R}$ such that $\phi = \{x^2 + 2x + 1 = 0\}$. A verification problem can be coded as a formula that can be checked for a solution using a satisfiability solver, this is the case for diagnosability of discrete event systems.

2.1.4 Boolean SAT

It is the particular case where ϕ is a formula over Boolean variables. So one asks whether the variables of a given Boolean formula can be consistently replaced by

the values TRUE or FALSE in such a way that the formula evaluates to TRUE. If this is the case, the formula is called satisfiable. On the other hand, if no such assignment exists, the function expressed by the formula is identically FALSE for all possible variable assignments and the formula is unsatisfiable. Example: The formula "a AND NOT b" is satisfiable because one can find the values a = TRUE and b = FALSE, which make the formula TRUE. In contrast, "a AND NOT a" is unsatisfiable.

Definition 2.1 (K-Sat Problem)**.** *Preliminary definitions*

1. *Atom: Boolean variable which can be either true or false, e.g.,* x, y
2. *Literal: an atom or its negation, e.g.,* x, $\neg x$
3. *Clause: a disjunction of literals, e.g.,* $x \vee \neg y$
4. *CNF: a formula in Conjunctive Normal Form consists of ANDs of several clauses, e.g.,* $(x \vee \neg y) \wedge (x \vee y)$

K-SAT problem is : given a CNF formula f*, in which each clause has exactly* K *literals, decide whether or not* f *is satisfiable. That is, whether there is an assignment to the atoms such that* f *evaluates to TRUE.*

2.1.5 Satisfiability modulo Theories (SMT) [54]

Satisfiability modulo Theories (SMT) solvers couple a SAT solver with a solver of a given theory T to check the satisfiability of logical formulas whose atoms are interpreted in T. An SMT formula is generally quantified, i.e., can be expressed using $\forall$-universal quantifiers and $\exists$-existential quantifiers. For example, diagnosability verification of a timed automaton can be coded as a SMT problem. In particular, Satisfiability modulo ODEs solvers are suitable to reason about continuous and more generally hybrid systems. Many mathematical problems involving ODEs can be expressed as quantified SMT formulas, such as the following.
General Initial Value Problem Given a system of ODEs $\vec{x} = f(t, \vec{x}(t))$ and a point $(t_0, \vec{x}_0)$ in the domain of f, find a function $\vec{x}$ that is a solution to the ODE system and satisfies $\vec{x}(t_0) = \vec{x}_0$.
Some widely used SMT solvers are `Z3` [41], `CVC` [10], `OpenMath`.

2.2 Qualitative Modeling and Simulation

In this chapter, a summary of the principles of qualitative reasoning, modeling and simulation is presented as they will form a basis for the abstraction process considered in this thesis work. First, the basic concepts of qualitative reasoning are presented then the most important works accomplished in qualitative modeling and simulation are reviewed and discussed [78, 49, 86].

2.2.1 Qualitative Reasoning and Simulation

Intuitive concept Given an object or a concept, qualitative reasoning (or QR) refers to the usage of "quality" descriptors to illustrate it. For example, to characterize the mass of an object, one can reason numerically by attributing a real value for it under a certain precision metric. On the other hand, one can reason qualitatively: instead of giving a precise numerical value for the mass of an object, we point out that it is heavy. Consequently, the values that can be taken by the mass of the object which, when represented by real numbers, are infinite, are assigned a label from a finite qualitative set, e.g., $Qal = \{\text{“}light''\text{, “}heavy''\}$.

Quantitative mapped to Qualitative The qualitative labels are abstractions of numerical valuations, which implies naturally that a relation between both can be defined. Intuitively, a modeling approach that uses numerical procedures always considers (up to a certain sufficient precision) that the attributed numerical valuations to the model variables are a faithful and sufficient representation of reality. A qualitative modeling approach describes the valuations using ranges. Back to our simple object mass example, one can map numerical values to their qualitative sets. We define a mapping function $\alpha : \mathbb{R}_+ \rightarrow Qal$ by, e.g., $\alpha(x) =$ “$light''$ if $x \in [0, 100]$ and $\alpha(x) =$ “$heavy''$ if $x \in (100, +\infty)$.

Safe modeling of partially known systems If one is told this object is “$heavy''$, one cannot point back the numerical value of the mass of the object. This implies the use of the partially provided information to reason about the object, with nevertheless frequently important decisions to make. Consider a robotic lifter arm whose maximum weight lifting capacity is of 100. The arm, provided the

correct qualitative label assigned by α, can decide whether or not to lift the object without any further information. We remind next some of the main applications of QR (which are not limited to).

Application Domain Qualitative reasoning has been applied to numerous domains, such as robotics, computer science, formal methods and verification, modeling of cyber physical-systems, testing and prototyping, biology and so on. The usage of QR tackles the fundamental issue of states explosion when simulating systems with large number of dependent entities and when numerical simulations are not rigorous enough to validate the system. Qualitative reasoning is mainly present in, but not limited to:

- Artificial intelligence: this is one of the main domains where qualitative reasoning is used, as real time decision making processes require a fast recognition of the context in which to operate and do not necessarily require a high level of precision.
- Embedded Control and Signal Processing: computer programs are limited in computational resources and qualitative reasoning can provide information at the sufficient abstract level to make correct processing and controlling.
- Economics: specific data are unknown most of the time, but the changing behavior, such as prices fluctuation of stock markets, is known. Thus, qualitative reasoning can be used to model the partially known system and perform a simulation with the given information and "predict" the possible future fluctuations.

2.2.2 Qualitative models of continuous systems

In this section, we introduce the mathematical foundations of qualitative reasoning and some of the main works done in this field. As a start, we consider continuous systems modeled as a set of functions and we study qualitative models assigned to these systems.

Continuity and Differentiability Reminders [f is continuously differentiable] is equivalent to [f is differentiable (hence continuous) and $f'(x)$ is continuous] equivalent to the notation [$f \in C^1$]. A critical point of a differentiable function f is a point where the derivative of f is null. Let $\overline{\mathbb{R}} = \mathbb{R} \cup \{+\infty\} \cup \{-\infty\}$.

Definition 2.2 (Reasonable Function)**.** *A function $f : [a, b] \subseteq \overline{\mathbb{R}} \to \overline{\mathbb{R}}$ is* reasonable *if all of the following are met: f is continuous on $[a, b]$; f is continuously differentiable on (a, b) with values in $\mathbb{R}$ and $f'_d(a)$ and $f'_g(b)$ exist in $\overline{\mathbb{R}}$ (when defined, i.e., when a or $f(a)$ is finite and the same with b); the set of critical points of f in any bounded interval of $[a, b]$ has only finitely many connected components; both limits $lim_{t\to a^+} f'(t)$ and $lim_{t\to b^-} f'(t)$ exist in $\overline{\mathbb{R}}$ and are equal respectively to $f'_d(a)$ and $f'_g(b)$ when defined.*

Assumption 2.1. *All functions modeling the constraints in the qualitative models we will consider in this section are assumed reasonable.*

Landmarks of continuous functions Let $S = \{f_1, ..., f_n\}$ be a set of functions of one variable t defined on $[a, b]$. We will refer to S by complete model and to t by time. One of the first and most interesting works in qualitative reasoning is the identification of important points in a function evolution, called *landmarks* [74]. The aim is to keep less information about the infinite $f_i(t)$ values but which is nonetheless characterizing what is "important" about S. To each function f_i let us assign a finite set of landmarks in $f_i([a, b])$, this set of points forms the basis of the qualitative behavior description of S and contains at least the functions critical points (where $f'_i(t) = 0$) when the f_i's are differentiable over their domain. The time points at which these landmarks occur are called *Time-Distinguished Points* (TDP). Note that the set $L(f)$ of landmarks of a function f is an ordered set.
Example. Figure 2.1 is a plot illustrating a function with four landmarks over the viewed interval. The TDPs are represented in green.

Definition 2.3 (Qualitative Attribution Function)**.** *Let f be a continuous function over a domain $X \subset \mathbb{R}$ and L a set of landmarks of f. We define the qualitative attribution function $Qf : X \to L \times L \times SIGN$, where $SIGN = \{+, -, 0\}$, as assigning to each real $x \in X$, first the two successive landmarks l_i and l_{i+1} of f*

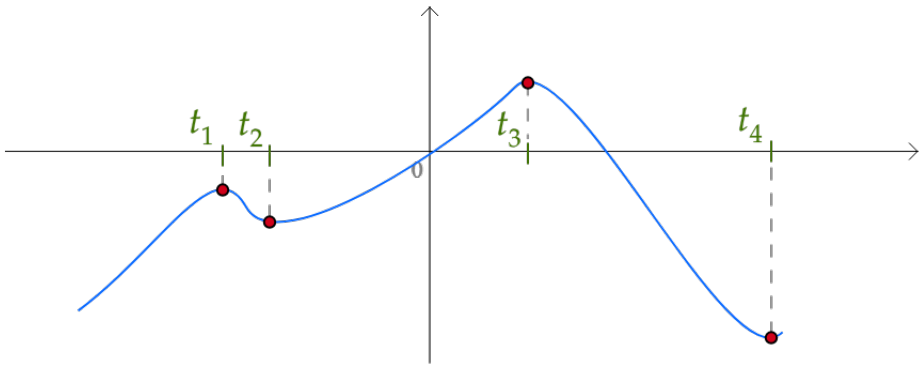


Figure 2.1: Graph of a function labeled with the minimum set of landmarks and time-distinguished points

such that $l_i < f(x) < l_{i+1}$ *if* x *is not a TDP and two times the landmark* l_i *if* $f(x) = l_i$*, and second the sign of* $f'(x)$*.*

Qualitative States Given a function $f : [a, b] \to \mathbb{R}$ where $a, b \in \mathbb{R}$ and a set L of landmarks of f with Qf the qualitative attribution function, then each attributed value by Qf is a qualitative state, the set of which will be denoted by QS.

Definition 2.4 (Qualitative State)**.** *Let* $l_1 < ... < l_k$ *be the landmark values of* $f : [a, b] \to \overline{\mathbb{R}}$*. The qualitative state* $QS(f, t)$*, for* $t \in [a, b]$*, is a couple* (q_{val}, q_{dir})*. For a given real* $t_0 \in [a, b]$ *then* $QS(f, t_0)$ *is given by:*

- q_{val} *(qualitative state value) is the landmark value* $f(t_0)$ *if* t_0 *is a TDP, or the couple of the previous and next landmarks* (l_j, l_{j+1}) *encompassing* $f(t_0)$ *if* t_0 *is not a TDP.*
- q_{dir} *(qualitative state direction) is* $+$ *if* $f'(t_0) > 0$*,* 0 *if* $f'(t_0) = 0$ *and* $-$ *if* $f'(t_0) < 0$*.*

Example 2.1 (Continuous Square Function)**.** *Consider* $f(x) = x^2$*. One of its landmark is* 0 *for the TDP* $x = 0$*. So, at the coarsest level, there are three qualitative states* $S_1 = ((-\infty, 0), -)$*,* $S_2 = (0, 0)$*,* $S_3 = ((0, +\infty), +)$*. The number of qualitative states grows linearly with the number of landmarks.*

Definition 2.5 (Qualitative Behavior)**.** *The qualitative behavior of f on $[a, b]$ is the sequence of qualitative states obtained by alternating timed-distinguished points of f and open intervals between two such consecutive points: $QS(f, t_0), QS(f, (t_0, t_1))$, $QS(f, t_1), ..., QS(f, t_n)$ such that, for all i, $QS(f, (t_i, t_{i+1})) = (\bigcup_{t_i<t<t_{i+1}} q^t val, q_{dir})$ with $(q^t_{val}, q_{dir}) = QS(f, t)$.*

2.2.3 Qualitative Differential Equations (QDEs)

Ordinary differential equations An ordinary differential equation (ODE) describes the relation between real variables and their change w.r.t time. In other words, in dimension two for example, for a given time value t_0 for which the values of x and y are x_0 and y_0, an ODE represents how the future (or past) values of the variables (i.e., after or before some infinitesimal time δt) are obtained according to the current values x_0 and y_0 and perhaps the time moment t_0 as well. This can be described as :

$$x(t_0 + \delta t) = f_1(x_0, y_0, t_0)\delta t \tag{2.1}$$

$$y(t_0 + \delta t) = f_2(x_0, y_0, t_0)\delta t \tag{2.2}$$

To model the variables change continuously, δt is supposed infinitesimally small, hence the usual representation of an ODE $\dot{\mathbf{x}} = f(\mathbf{x}, t)$. ODEs are widely used to model systems with tightly dependent variables changing continuously with time.

Qualitative differential equations A qualitative differential equation (QDE) also constrains the future evolution of the variables according to their past values. However, a QDE relaxes the constraints of the ODE, it is thus an abstraction of the ODE. The main difference between a QDE and an ODE is that the valuations of the variables belong to a finite domain instead of the usual $\mathbb{R}^n$. This finite domain is defined by given landmarks associated to time-distinguished points of a function. For example consider the differential equation $\dot{x} = x + 1$ assigning to each real value of $x \in \mathbb{R}_+$ a value of the derivative $\dot{x}$ with respect to time. Suppose x is a traveling distance and $\dot{x}$ is the traveling speed. And that, according to a specification, different actions are to be performed for each different speed interval from $Qal = \{$"*slow*", "*normal*", "$fast$"$\}$, where "*slow*", "*normal*" and

"$fast$" label respectively the sets $\dot{x} \in [1, 50]$, $(50, 100]$, and $(100, +\infty)$. Given the partitioning Qal of $\dot{x}$ we can define a qualitative differential equation as a mapping relating regions of x to regions of $\dot{x}$ as:

$$\Delta X : Q \to Qal \tag{2.3}$$

where $Q = \{q_{slow}, q_{normal}, q_{fast}\}$ such that: $q_{slow} = [0, 49]$, $q_{normal} = (49,\ 99]$ and $q_{fast} = (99, +\infty)$, and $\Delta X(q_{slow}) = slow, \Delta X(q_{normal}) = normal, \Delta X(q_{fast}) = fast$.

2.2.4 Using automata to model qualitative differential equations

Definition 2.6 (Automaton)**.** *An automaton A is a tuple $A = (Q, \Sigma, T, Q_0, Q_F)$ where:*

- *Q is a set of states*
- *Σ (or alphabet) is a finite set of labels (or symbols)*
- *$T \subseteq Q \times \Sigma \times Q$ is a set of labeled transitions*
- *$Q_0 \subseteq Q$ and $Q_F \subseteq Q$ are sets of respectively initial and final (also called accepting) states*

An execution (or run) r of A is a sequence $l_0 l_1 ... l_n$ where $l_i \in Q$, $l_0 \in Q_0$ and for each consecutive states $l_i l_{i+1}$ there is a label $\sigma \in \Sigma$ such that $(l_i, \sigma, l_{i+1}) \in T$. If additionally $l_n \in Q_F$, then r is an accepting execution.

We can model a qualitative differential equation using an automaton. For this purpose, let $A = (Q, T)$ be a simple automaton where Q is a set of states and T is a set of transitions $T \subseteq Q \times Q$. For our previous example, we consider each region of x as a state: q_{slow}, q_{normal} and q_{fast}. A transition $t = (q_1, q_2) \in T$ represents that there could be (but not necessarily) at least one trajectory initially in q_1 that reaches, after a certain time, q_2 (without reaching any other intermediate state between). For the state q_{slow} this means two outgoing transitions $t_1 = (q_{slow}, q_{normal})$, $t_2 = (q_{slow}, q_{slow})$. Given only the two pieces of information $x \in [0, 49]$ and

$\dot{x} \in [1, 50]$, we can conclude that an outgoing transition is guaranteed to an adjacent region, i.e., t_2. Given an automaton A, then if $\exists q \in Q, t = (q, q) \notin T$ then the region of q is a singleton and $0 \notin \Delta X(q)$. In other words, we can relate any differentiable trajectory to a path of the automaton derived from the qualitative differential equation, however the inverse is not true. The constructed automaton provides a computational model over which model checking methods are applied to obtain verification results on the system. In our case, the automaton can provide non reachability results, i.e., deciding if some regions of the state space are never reached. Starting from a set of initial regions, a region is never reached if there is no path rooted in the set of initial states and leading to it in the corresponding automaton.

Example 2.2 (U-shaped tube)**.** ***Modeling with Qualitative Reasoning***

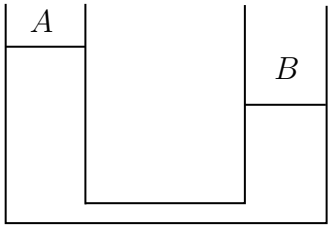


Figure 2.2: Two tanks system

The presented example is a simplified version taken from QSIM encyclopedia consisting of a U-shaped tube that shows a practical case to model a system using qualitative principles [73]. Water is filled with two different levels A and B and a thin tube linking both of them (Figure 2.2). Water is added to tank A, which causes a pressure difference between the two tanks, water passes through the thin tube until both tanks have the same water level. The problem is studied using the variables: H_A, H_B water quantities (proportional to water levels) respectively in tanks A and B; P_A and P_B pressures at the bottom of tanks A and B respectively. The variables are continuous in time. The pressure in the tank increases with the amount of water, this constraint can be modeled qualitatively by $P = g^+(H)$ with g^+ some monotonically increasing continuous function. Let us associate landmarks to the model variables: to H_A we associate $L_{H_A} = \{0, H_{A_{MAX}}, +\infty\}$ where $H_{A_{MAX}}$

is the maximum water quantity in tank A and to P_A we associate $L_{P_A} = \{0, +\infty\}$. Similar qualitative constraints are assigned to variables H_B and P_B. Additionally we know about the function g^+ that, if the tank is empty, then the pressure is null and analogically, infinite water quantity causes infinite pressure: thus we express these two statements with regards to the landmarks as $g^+(0) = 0$ and $g^+(+\infty) = +\infty$. The flow $flowAB$ in the thin tube (i.e., the quantity of water going through a section of the tube during a time unit) results from a pressure difference in the two tanks $p_{AB} = p_A - p_B$, what we model by another monotonically increasing function f^+ such that $flowAB = f^+(p_{AB})$ with $f^+(0) = 0$ and $f^+(+\infty) = +\infty$. If the water quantity in A decreases then the quantity in B increases accordingly and vice versa, thus we have $\frac{dH_B}{dt} = -\frac{dH_A}{dt}$. The flow is equal to this water quantity variation: $flowAB = \frac{dH_B}{dt} = -\frac{dH_A}{dt}$. The equations are qualitative differential equations where each variable V is not in $\mathbb{R}$ but in the finite domain $L(V)$ defined by the associated landmarks. The qualitative model is the set of all the constraints given previously (see Figure 2.3). In the next section, we recall the principles of simulating a given qualitative model.

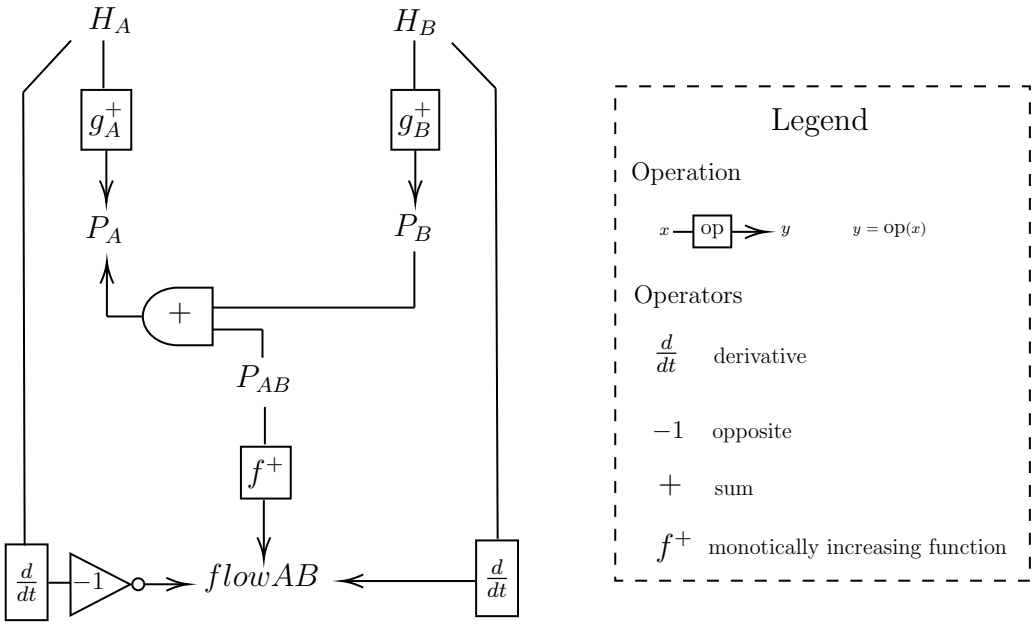


Figure 2.3: Qualitative model of the U-Tube example

2.2.5 Qualitative Simulation

After the modeling phase, one can perform a qualitative simulation that derives a set of possible behaviors from the model. Given an (or several) initial qualitative state, the simulation computes successors of it. Once a fixed point has been reached, i.e., the successors are not new qualitative states, the simulation stops. Since the qualitative space is finite when the considered functions are reasonable, termination of the qualitative simulation algorithm is guaranteed. The successor is computed by applying a set of rules relying on the continuity and differentiability assumptions. The simulation results can be used to verify if specifications or desired properties are met, i.e., for model checking purposes [74]. Adaptive reasoning can be incorporated to the simulation, whereas some significant points can be identified and used in the simulation, increasing the size of the **landmarks** set and thus the accuracy of the simulation (the number of dynamically created landmarks has to be finitely bounded to guarantee termination of the simulation). Compared to numerical approaches, qualitative simulation allows the usage of only partial information provided about the system (derivative sign, curvature points, other important points) and termination of the simulation is guaranteed since the state space is abstracted to a finite space. On the other hand, one looses the accuracy that is provided by a numerical approach. The fundamental steps are:

- Find the landmarks of all continuous functions representing the system.
- Find all qualitative states whose landmarks cover the given initial set. Mark these states.
- Compute successor qualitative states of each marked state by applying the qualitative rules and unmark it.
- Stop the computation when all states are unmarked.

The result is a graph where each path is a sequence of qualitative states $(QS_0, ..., QS_n)$ where QS_{i+1} is a successor of QS_i. Every change from one qualitative state to another is associated to a time-distinguished point. The graph illustrates possible behaviors of the system and can be analyzed to verify given specifications.

Example 2.3 (U-shaped tube)**.** ***Qualitative Simulation*** The simulation provides a set of the different possible behaviors of the system. Back to the U-tube example, we know that from any initial state, the levels in the two tanks will converge to become equal and the flow $flowAB$ to become null. Consider an initial state where tank A is full and B is empty, i.e., $H_A = H_{A_{max}}$ and $H_B = 0$. Let us find the initial qualitative states for the rest of the variables together with their derivative signs. By inference rules, $P_B = 0$ and $P_A = (0, +\infty)$, consequently

Qualitative Variable	q_{val}	q_{dir}
H_A	$H_{A_{max}}$	$-$
H_B	0	$+$
P_A	$(0, +\infty)$	$-$
P_B	0	$+$
P_{AB}	$(0, +\infty)$	$-$
$flowAB$	$(0, +\infty)$	$-$

Table 2.1: Initial qualitative state

$p_{AB} = (0, +\infty)$. Similarly $flowAB = (0, +\infty)$ which determines that the flow is from A to B, i.e., $qdir(H_A) = -$ and $qdir(H_B) = +$. g_A^+, g_B^+ and f^+ are monotonically increasing functions, which implies that $qdir(P_A) = -$ and $qdir(P_B) = +$ and then $qdir(P_{AB}) = -$ and finally $qdir(flow_{AB}) = -$. This completes all valuations of the initial qualitative state $QS(t = 0)$ (Table 2.1). Let us compute the successor states of $QS(t = 0)$, i.e., the the new possible valuations of the qualitative variables at a time $t > t_0$. H_B is initially zero and increasing, since the increase process is continuous thus in the next state H_B will be positive and still increasing. Similar reasoning is applied to the rest of the variables to obtain the successor qualitative state (Table 2.2). The applied rules rely on the continuously differentiable assumption for each variable. The successor state is valid for the open time interval (t_0, t_1) where t_1 is the next time-distinguished point.

At t_1, different qualitative states are possible. Either an equilibrium is reached with $flowAB$ being null, or $flowAB$ continues to decrease. In the first case, there are two possibilities, either the tank B is filled at the same time the equilibrium is reached (QS_1, Table 2.3), or B is only partially filled while reaching the equilibrium (QS_2, Table 2.3). In the second case, tank B is fully filled but the equilibrium is

Qualitative Variable	q_{val}	q_{dir}
H_A	$(0, H_{A_{max}})$	$-$
H_B	$(0, H_{B_{max}})$	$+$
P_A	$(0, +\infty)$	$-$
P_B	$(0, +\infty)$	$+$
P_{AB}	$(0, +\infty)$	$-$
$flowAB$	$(0, +\infty)$	$-$

Table 2.2: Successor qualitative state

still not reached, thus the current model can represent the spilling effect of water from tank B (QS_3, Table 2.3). Thus, many next qualitative states are possible, the transition system obtained from a qualitative simulation is not deterministic.

Qualitative Variable	Branched Qualitative States					
	QS_1		QS_2		QS_3	
	q_{val}	q_{dir}	q_{val}	q_{dir}	q_{val}	q_{dir}
H_A	$(0, H_{A_{max}})$	0	$(0, H_{A_{max}})$	0	$(0, H_{A_{max}})$	$-$
H_B	$H_{B_{max}}$	0	$(0, H_{B_{max}})$	0	$H_{B_{max}}$	$+$
P_A	$(0, +\infty)$	0	$(0, +\infty)$	0	$(0, +\infty)$	$-$
P_B	$(0, +\infty)$	0	$(0, +\infty)$	0	$(0, +\infty)$	$+$
P_{AB}	0	0	0	0	$(0, +\infty)$	$-$
$flowAB$	0	0	0	0	$(0, +\infty)$	$-$

Table 2.3: Branching of successor qualitative states

2.3 Hybrid Systems Verification

This section reviews the literature concerned with verification methods for hybrid systems and more specifically reachability analysis. In the existing literature, we distinguish several modeling frameworks for hybrid systems: hybrid automata, Petri nets, hybrid bond graphs and hybrid programs. In this thesis, we adopt hybrid automata as models of hybrid systems because of their common use in the scientific community and their intuitive way of coupling finite state machines with differential equations. A formal definition of a hybrid automaton and its semantics are proposed in chapter 3 and will be used for the following sections (Def. 3.1,

p.53 and Def. 3.2, p.57). For the following, let H be a hybrid automaton and $[[H]]$, the set of all executions of H (i.e., the semantics of H).

2.3.1 Hybrid Automata Reachability Analysis

Reachability analysis for hybrid systems received a considerable attention in the literature due to the emerging need for formal verification of complex critical systems, in particular of cyber-physical systems.

Reachability Problem Statement Let B be a set of states. We wish to verify if any execution of H starting from its initial set reaches B at some time. Determining the reachable set of states with certainty decides whether or not the system is safe. However, for hybrid systems, reachability analysis is generally undecidable. Much work has been devoted to verify state reachability through the computation of an over-approximation containing the concrete executions $[[H]]$. Another emerging research direction is the study of under-approximations of $[[H]]$, a problem that is tackled in less work.

We summarize and review three large categories of hybrid automata verification techniques: numerical simulations, over(under)-approximation flow-pipes and invariant synthesis, and symbolic abstraction techniques.

2.3.2 Numerical Simulation

Let $C = (X, S_0, F, Inv)$ be a continuous system, where the dynamics F is given by Lipschitz continuous functions w.r.t. all variables in X. Given an element x_0 from the set S_0 then numerical simulation consists in computing a trajectory $\phi : T \to \mathbb{R}$ at some time points $T \subseteq \mathbb{R}_+$ starting from the initial element $\phi(t_0) = x_0$. This trajectory is unique for a given x_0 as stated by the *Picard-Lindelof theorem*. Numerical simulation explores the reachable set of states depth first. This method examines trajectories one by one. The exact trajectory is abstracted to a set of points computed at specific time moments via the integration step Δt. To validate a model via numerical simulations, a number of them are performed and when each of them does not violate the safety property then the system is supposed safe up

to the considered precision (estimated from the integration step and the tolerated approximation error). Numerical simulations are aimed at local verification given some bounded local error and are not suitable for studying a modeled perturbation. Today, numerical simulation is widely used to validate and test models in the industry. Many schemes have been elaborated for numerical simulations such as Runge-Kutta and Euler's method. *Matlab & Simulink* offers a wide library of models applied in diverse fields and is of common use for academics and industry. The simulation part in the software is limited to numerical simulation. The simulation algorithm such as *ode45* is sophisticated and is able to use shortcuts in some situations in an ad hoc manner for increased computation efficiency or precision. In critical industry applications notably Avionics, the certification process remains today time consuming and often based on documentation. Consequently, there is an emerging urgent need for tools with better formal basis, where the reached precision is accurately quantified. Set-based simulation consists in computing the reachable states from the whole initial set S_0, this is a breadth first search and will be reviewed in the next paragraph. The Acumen simulator applies numerical simulation while having solid underlying semantics [100], it offers a good understanding of the underlying computations in such a way that the user is directly confronted to problems that could arise such as the zero-crossing problem. The latter problem occurs when the chosen precision is not enough to test whether or not the computed trajectory satisfies a given condition.

2.3.3 Flow-pipe methods

The term flow-pipe has been coined as an over-approximation set of reachable states of a continuous dynamical or hybrid system given an initial set of states [25, 79, 75, 9]. In the next chapter we will review some tools that compute the flow-pipe of the executions of a hybrid system from the initial set of states. These tools showed success in verifying safety and robustness in reaction to perturbations or uncertainty. The mathematical foundation of the tools relies on set based integration and safe approximations of functions. **Intuitively**, the idea is the following: given the initial set and the differential equations attributing locally the first order derivative of each variable, a safe upper bound of the first order

derivative is computed. Then, given a time step, the computed bounds derive a constraint over the variables valuations, representing parts of the state space that cannot be reached at this time step. For example, a car is at distance $d = 0$ and running at $2km/min$ initially at $t = 0$. A safe lower and upper bounds of 2 are 1 and 3, the car is supposed never to over-or-underpass these bounds. At $t = 5min$, we can safely say that the car is at a distance of at least $5km$ and not exceeding $15km$. In general, the exact first derivative is not constant but it is continuously changing according to the variables of the state space and possibly time. Consequently, the bounds over the dynamics are changing with time and must be computed accordingly..

Computing a flow-pipe Algorithm 1 presents a rough scheme for computing a flow-pipe with some notation abuse.

input : $H :=$ Hybrid Automaton;
$\Delta t :=$ Time Step (float precision real);
$T :=$ Time Horizon (float precision real)
output: Flow-pipe $:= Flow(H)$

$Flow \leftarrow \{(q_0, Init(q_0)\}$;
$Time \leftarrow 0$;
$List \leftarrow Flow$;
while $List \neq \emptyset \wedge Time < T$ **do**
  $(q, S) \leftarrow List.pop()$;
  $C \leftarrow$ `ContinuousTimeElapsed` $(S, \Delta t)$;
  $Time \leftarrow Time + \Delta t$;
  $Flow \leftarrow Flow \cup \{(q, C)\}$;
  for G *guard in* q **do**
    $D \leftarrow$ `ComputeDiscreteJump` (C, G);
    if $D \neq \emptyset$ **then** $List \leftarrow List \cup \{D\}$;
  end
end
return $Flow$;

Algorithm 1: Hybrid automaton flow-pipe computation rough scheme

Algorithm operations The algorithm does not present all the required details, for example one needs to remove from $Flow$ the parts that do not satisfy the invari-

ant of each mode but this operation has been omitted. It is written here for only one initial mode. The reachability algorithm requires operations over the states S, represented each one by a subset of $\mathbb{R}^n$. The main procedures that are required are:

• `ContinuousTimeElapsed` $(S, \Delta t)$: given a state S reached at some time t and the time step Δt, the procedure computes a continuous successor of S denoted S' reached at $t + \Delta t$. All the concrete trajectories of H initially in S are in S' after Δt time units have elapsed. The procedure is the challenging part in reachability computation because it requires using the dynamics of H to extract S'.

• `ContinuousDiscreteJump` (S_1, S_2): given two states S_1 and S_2 in a mode, where S_2 corresponds to the change of mode condition associated to a transition from this mode (also called guard/jump condition), this operation computes the reachable set via the discrete jump. In other words, an intersection between the guard and the current reached set is computed then its image via the reset (if present) is computed.

Additionally, the regular set based operations are used by the reachability algorithm such as: Minkowsky sum $\oplus$, Union $\cup$ and intersection $\cap$.

Forward and backward reachability Flow-pipe computation tackles straightforwardly safety verification, the time step for integration is always taken forward starting from the initial set. To incorporate the specification (i.e., the unsafe set), many tools compute two flow-pipes: the first one, $Flow(S_0, T)$ is initialized the initial set with a forward time integration step and the second $Flow(B, T')$ is initialized from the unsafe set with a backward integration time step (Figure 2.4). If the two flow-pipes do not intersect, the system is safe, if not, back and forth operations are applied to further refine the flow-pipes until a specified precision is achieved (i.e., computing more accurate bounds) [84]. Computing two flow-pipes instead of one reduces the computation time for the verification process by making use of parallelism.

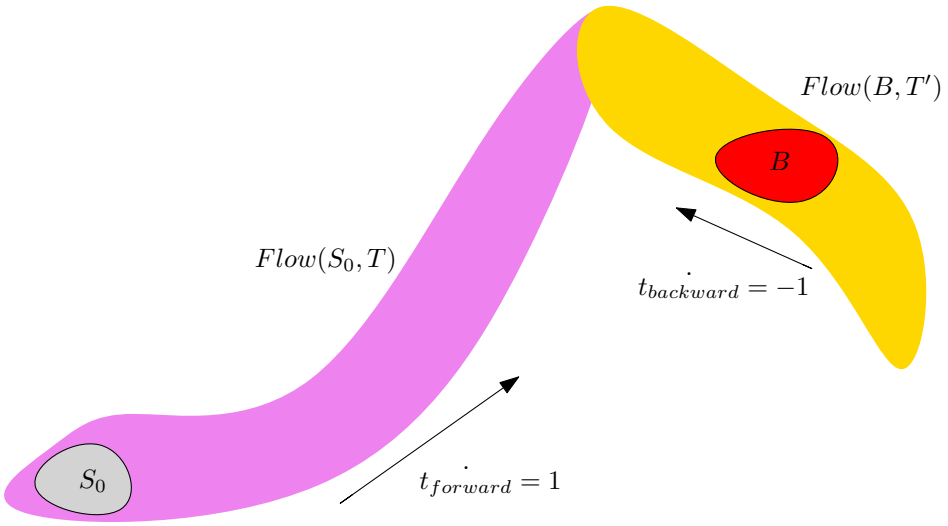


Figure 2.4: Forward and backward flow-pipe computation for safety verification

Geometric representation of states Hence, the time and memory complexities of the used operations depend crucially on the choice of the state representation S. Many works have been devoted for the choice of the state representation. An important concern in these methods is to balance the trade off between accuracy and computation complexity [79]. The need for accuracy is seen when computing the continuous time elapsed, where the error between the over-approximate representation and the actual concrete state should be bounded. The computation complexity arises when performing operations such as set addition (Minkowsky sum), intersection and union. We review important state representations.

□ **Convex sets** have been extensively studied as state representations.

Definition 2.7 (Convex Set)**.** *A set $S = \{\boldsymbol{x} \mid x \in \mathbb{R}^n\}$ is convex if and only if for all x and y in S and any real $\lambda \in (0, 1)$, $(1 - \lambda)x + \lambda y \in S$.*

A convex polyhedron is obtained by intersecting any number of half-spaces (Figure 2.5). A polytope is a bounded convex polyhedron.

Definition 2.8 (Hyper-rectangle)**.** *A hyper-rectangle HR of dimension n is a subset of $\mathbb{R}^n$ such that HR is a product of boxes: $HR = \prod_{i=1}^{n} I_i$ where each I_i is an interval over the reals.*

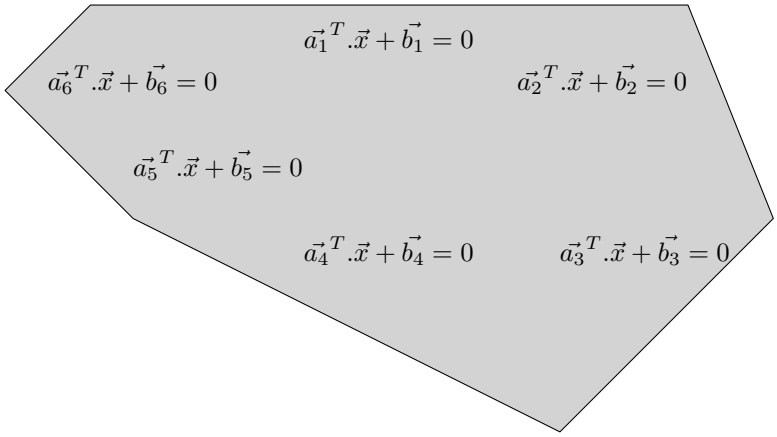


Figure 2.5: Convex polytope, intersection of half-spaces

Zonotopes

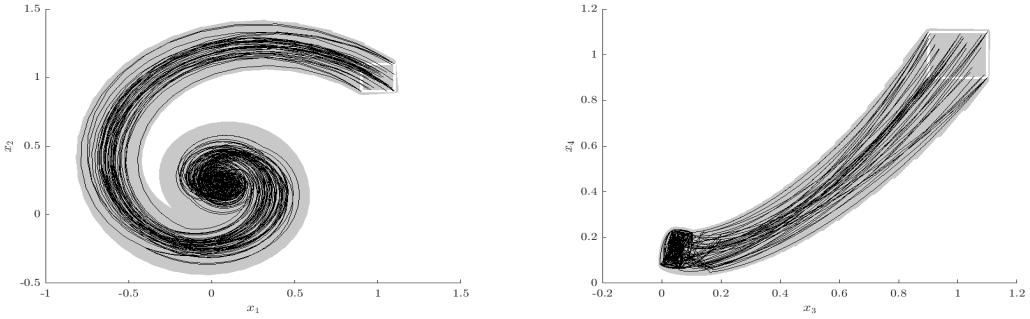
Definition 2.9 (Zonotope)**.** *A zonotope $Z = (G, c)$ where $G \in \mathbb{R}^{n\times m}$ is the generator vectors set matrix with $m, n \geq 0$, $c \in \mathbb{R}^n$ is called the center of Z is the set $\mathcal{Z} \subset \mathbb{R}^n$ such that:*

$$\mathcal{Z} = \{x \in \mathbb{R}^n \mid \exists a \in [-1, 1]^m, x = G.a + c\} \tag{2.4}$$

A zonotope is symmetrical w.r.t to its center c, a more generalized representation that is not necessarily symmetrical is given by star sets. A star set is the intersection of a zonotope with a set defined by some predicate constraints.

Example 2.4 (Flow-pipe computation using zonotopes)**.** Consider the following linear system where the initial set is given by a hyper-rectangle set $x(0)$ and the state $x(t)$ is subject to bounded perturbation given by a zonotope $u(t)$. We use *CORA*, a tools suite for reachability computation of continuous and hybrid systems, to compute the flow-pipe of the system using zonotopes (Figure 2.6). The flow-pipe (in grey) is computed till the time horizon $T = 5$ time units. *Matlab Simulink* is used to run numerical simulations randomly generated from the same initial set observed in black.

$$\dot{x} = \begin{bmatrix} -1 & -4 & 0 & 0 & 0 \\ 4 & -1 & 0 & 0 & 0 \\ 0 & 0 & -3 & 1 & 0 \\ 0 & 0 & -1 & -3 & 0 \\ 0 & 0 & 0 & 0 & -2 \end{bmatrix} x + u(t), \quad x(0) \in \begin{bmatrix} (0.9, 1.1) \\ (0.9, 1.1) \\ (0.9, 1.1) \\ (0.9, 1.1) \\ (0.9, 1.1) \end{bmatrix}, u(t) \in \begin{bmatrix} 0.9, 1.1 \\ (-0.25, 0.25) \\ (-0.1, 0.1) \\ (0.25, 0.75) \\ (-0.75, -0.25) \end{bmatrix}$$



(a) (b)

Figure 2.6: Flow-pipe computation versus numerical simulations

Support functions Support functions appeared efficient in computing over-approximations of convex sets. A support function *Supp* of a convex, compact set S is a mapping $Supp_S : \mathbb{R}^n \to \mathbb{R}$ such that $Supp_S(l) = max_{x \in S} \quad l.x$. Convex sets can be represented via their support functions. This representation showed efficiency in the computation of different operations needed for reachability computation in comparison with direct expression of the set via constraints. The study of support functions led to the development of the tool *SpaceEx* that will be discussed in the next section.

□ Non-convex sets

Taylor Models Taylor models are non convex sets that have been studied and applied as a representation of states for computing reachable sets. The Taylor model state representation is implemented in the tools *CORA* and *Flow**. Let f be a n time differentiable function such that $f : (a, b) \to \mathbb{R}$ where $(a, b) \subseteq \mathbb{R}$. Then we can say that:

$$f(x) = P_n(x) + R_n(x) \tag{2.5}$$

where for some $c \in (a, b)$

$$P_n(x) = f(c) + \frac{f'(c)}{1!}(x - c) + \frac{f''(c)}{2!}(x - c)^2 + ... + \frac{f^{(n)}(c)}{n!}(x - c)^n \tag{2.6}$$

And if $f^{(n+1)}$ exists and is continuous on an open interval containing c and x is in this interval, then there is some $d \in (x, c)$ between such that:

$$R_n(x) = \frac{f^{(n+1)}(d)}{(n+1)!}(x-c)^{n+1} \tag{2.7}$$

Definition 2.10 (Over-approximative Taylor model)**.** *Given a polynomial $p \in \mathbb{R}[X]$, where X is a set of n variables, an interval I, and a function f defined over a domain $U \subseteq \mathbb{R}^m$, then (p, I) is an over-approximative Taylor model of f if:*

$$\forall x \in U, f(x) \in p(x) + I \tag{2.8}$$

To construct an over-approximative Taylor model of a function f one can: compute till a certain order the Taylor polynomial of f, find the interval $I = (q, r)$ using the remainder.

$$q \leq R_n(x) \leq r \Rightarrow P_n(x) + q \leq P_n(x) + R_n(x) \leq P_n(x) + r \Rightarrow p_2(x) \leq f(x) \leq p_1(x) \tag{2.9}$$

Example 2.5 (Taylor model of e^x)**.** Consider $f(x) = e^x$ and $x \in (-1, 1)$. We use a method proposed in previous works to find a suitable Taylor model of $f(x)$ [24]. Compute the Taylor polynomial at the midpoint of $(-1, 1)$ till a certain order then evaluate a upper and lower bound of the remainder. Observing Table 2.4,

Polynomial Order k	P	I
0	1	$[-0.75, 1.75]$
1	$1 + x$	[0,0.75]
2	$1 + x + \frac{x^2}{2}$	[-0.25,0.25]
3	$1 + x + \frac{x^2}{2} + \frac{x^3}{6}$	[0.0345,0.0517]
4	$1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \frac{x^4}{4!}$	$[-0.02266, 0.02266]$

Table 2.4: Taylor polynomial of the exponential function with a safe remainder interval

we notice that the higher the computed Taylor polynomial order is, the lower the remainder error (given by I) is. If the computation of the remainder satisfies: $f(x) - P_k(x)$ converges to zero if $k \to +\infty$, then any arbitrary precision can be computed by increasing the Taylor polynomial order.

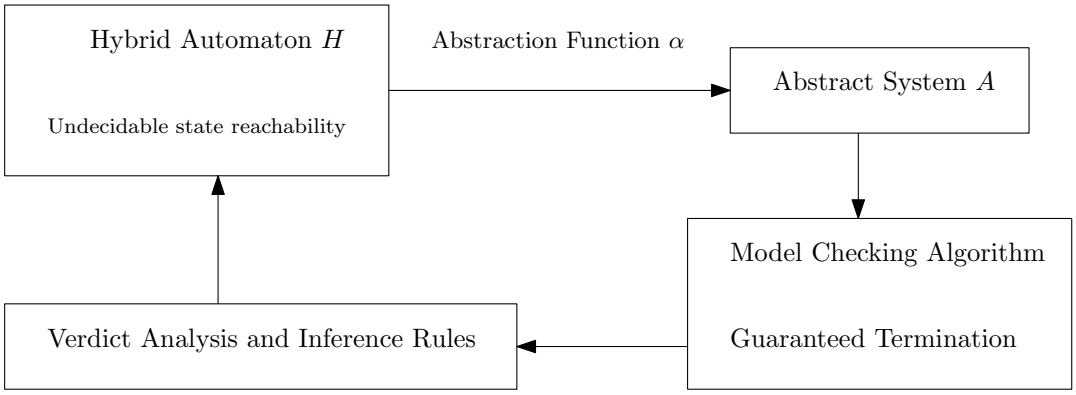


Figure 2.7: Hybrid automata abstraction scheme combined with model checking

2.3.4 Abstraction Techniques

Flow-pipe computation provides methods for accurately computing the reachable set, however termination of the algorithm cannot be generally guaranteed without specifying the time horizon T. In some situation, self-containment is achieved (i.e., there is some time instant $t < T$ for which the flow-pipe part obtained at time $t + \Delta t$ is totally contained within one or more of the flow-pipe parts computed at times before t). Consequently, flow-pipes remain a bounded model checking algorithm suitable for continuous dynamical and hybrid systems. In many applications, verifying time-unbounded properties is necessary, most notably in controller certification. Some techniques emerged for synthesizing invariants, i.e., sets that hold for any time t. The main idea is to find functions in the state space over which all change directions point inwards. We review in this part existing abstraction methods and invariant synthesis techniques.

Abstraction Methods Abstraction methods extract information from a given model to show a desired behavior or property. Figure 2.7 illustrates the high level scheme for abstracting a hybrid automaton H into an abstract system A via the abstraction function α. These methods retain important information from the hybrid system that are possibly "sufficient" to prove the property. These properties can range from reachability or liveness of a set of states to more complex ones. In the existing literature, authors studied and elaborated different abstraction methods of a given hybrid system model. Methods for constructing quality abstractions

cannot be automated for the broad class of hybrid automata. This is explained by the fact that many differential equations have no closed form solutions and it is thus difficult to define a general abstraction method that would apply for any continuous change. Two fundamental issues have to find a solution by any abstraction method:

• Describe formally the abstraction method M that produces the abstract system given the concrete one (and automate the construction operations as much as possible) and give the algorithmic steps to compute it.

• Give the necessary proofs that once the abstract system, constructed with the method M, verifies property $F(P)$ then the original hybrid system verifies P where F relates a concrete property to its abstract counterpart. Or more generally, deduce a class of properties that is preserved in the abstraction constructed using method M.

Other questions follow naturally the previous ones such as the time and space complexity of the abstraction algorithm, termination and worse execution time, application of the method in a distributed framework and so on. An example is the state reachability property which is undecidable for hybrid systems. Nonetheless, if the abstract system falls within the decidable class, then the abstract counterpart of the property can be verified using classical model checking algorithms. If the property is verified at the abstract level, the abstraction relation allows one to infer this result back to the original hybrid system.

Predicate Abstraction Hybrid systems are generally infinite state machines. Early work in [4, 5] shed the light on the completeness of abstracting infinite state machines into finite ones for linear dynamical hybrid systems. The infinite space of the hybrid system is abstracted into the finite possible valuations of a given set of linear predicates. A valuation represents concretely a convex polyhedron. From each obtained abstract state, transitions towards other abstract states are computed according to the continuous and discrete changes. To compute transitions outgoing from an abstract state A a flow-pipe is computed with the concrete set

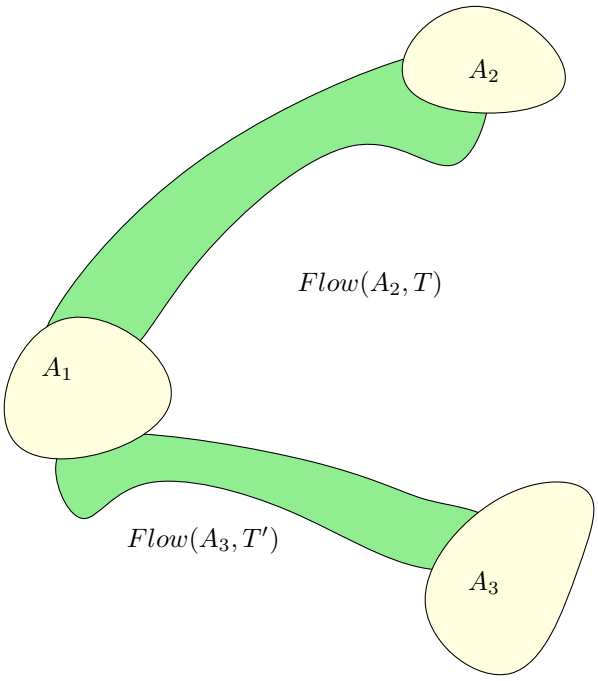


Figure 2.8: Three abstract states defined via predicates and linked via flow-pipe computation

represented by A and computed for time instants $t_1, t_2, ...$ After each step of the flow-pipe computation at time t_i, an intersection of the obtained flow-pipe with the concrete sets of abstract states different from A is performed. If the intersection with another state B is not empty, a transition is added from A to B. Flow-pipe computation uses existing tools for linear dynamics. Obviously, a maximum time horizon of the instants t_i must be provided and the number of performed intersections must be manually tweaked. Figure 2.8 illustrates three abstract sets A_1, A_2 and A_3 and the incoming abstract transitions towards A_1 obtained via flow-pipe computation. The idea of predicate abstraction came originally from software verification techniques. It showed effectiveness in finding bugs in programs. In [61, 102, 8, 96], Tiwari proposed to combine predicate abstraction with qualitative reasoning to have a more representative abstraction; his work is of an important basis for this thesis. A study for timing the abstraction is given in a complementary article [85]. In [99], Sloth addresses abstractions of non-linear systems with polynomial dynamics. The idea is to generate a partitioning of the state space,

whose abstraction is always complete and in some cases sound. For soundness, the task is formulated as an optimization problem that could yield results similar to those obtained by invariant synthesis techniques. Nonetheless, the termination of the optimization procedure is generally not guaranteed. [28] combines in a recent work relational abstractions and flow-pipe constructions allowing one to cover a larger class of properties, and shows that using each method separately fails to do so.

Invariant Synthesis Seen from a computational perspective, invariant computation consists in computing a set of mathematical constraints satisfied by all states of the system. The property we wish to prove is expressed as a set of constraints as well. One can then check if the property constraints violate any of the computed invariant constraints, if none is violated then the system verifies this property. Invariant computation is difficult and not automatic as the form of the invariant varies differently with the form of the dynamics and other components of the hybrid system. Some techniques assume the invariant takes a template form. For example, for a linear invariant with two real variables x, y, the form can be $\alpha x + \beta y = 0$ with α and β two unknown constants. Then one encodes the hybrid system, the property to verify and a specific parametric form of the invariant into an SMT (Satisfiability Modulo Theories) based solver and evaluates the unknown parameters α and β of the invariant automatically [61]. Other techniques attempt to generate barrier certificate functions that enclose the reachable set of states. A barrier certificate is an inductive invariant that helps in safety verification of dynamical and hybrid systems. For a given system, if a barrier certificate function is found and does not intersect the unwanted set of states, the system is considered safe and that is valid for any execution within the barrier certificate and for unbounded time (Figure 2.9). The main challenge is to find barrier certificates that can be efficiently encoded (i.e., expressed in a usable form), as their structure can become complex [71, 47]. Recently, generating barrier certificates was applied to safety verification in a compositional framework [98].

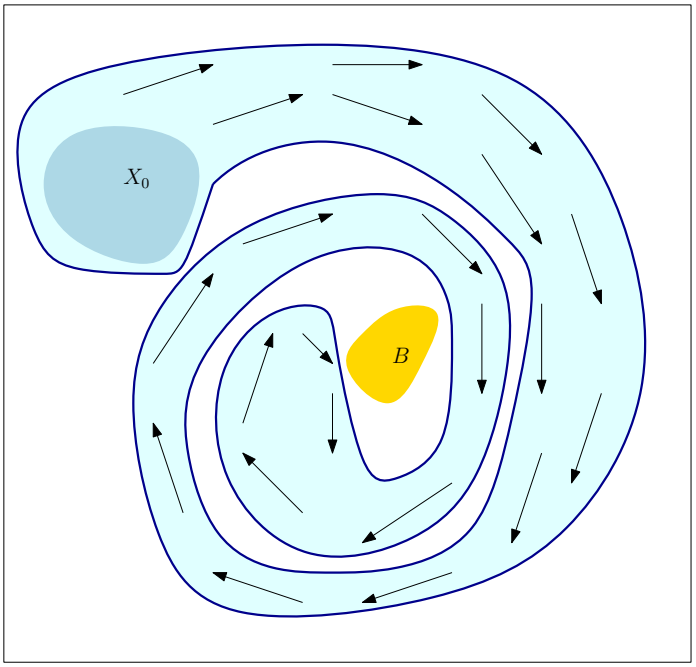


Figure 2.9: A barrier certificate enclosing the reachable set of states from X_0 without intersecting the unwanted set of states B

2.3.5 Counter-Example Guided Abstraction Refinement (CEGAR)

The CEGAR loop consists in combining the abstraction model checker together with a refinement method of the abstraction. Since model-checking can be inconclusive (i.e., when the abstraction is not sufficient to establish the proof), one can refine the abstraction for further precision. We now recall counter-example guided abstraction refinement (CEGAR) that was originally proposed to verify safety properties [48, 6]. Very generally, CEGAR is an incremental way to decide if a problem admits a solution. The method starts with very coarse abstraction of the original problem, checks the abstraction for a solution, then depending on whether or not a solution is found concludes that the original problem is solvable or that the abstraction does not retain enough information to make a decision. In the latter case a refinement is applied (i.e., information is added to the abstraction from the original problem) and the verification procedure is repeated on the refined abstraction.

As example, consider in our case the following situation, which will be made precise in the next chapters. Given a hybrid automaton H, a set $B \subset S$ of so-called "bad" states, where $S = Q \times \mathbf{X}$ is the state space of H, we say that H is safe with respect to B if and only if there is no execution in H from an initial state to a bad state in B. Otherwise, we say that H is not safe. This safety property is thus expressed as $H \models \mathbf{AG}\neg B$ in Computation Tree Logic (CTL), which means that all hybrid automaton executions never reach a bad state from B. An execution $h = (q_0, \mathbf{x}_0) \xrightarrow{l_0} (q_1, \mathbf{x}_1)...(q_m, \mathbf{x}_m)$ with $(q_m, \mathbf{x}_m) \in B$ is called a counterexample of H with respect to this safety property.
Let DA a discrete automaton obtained by applying an abstraction function α on H. We say that $\hat{h} = \hat{s_0} \xrightarrow{\hat{l_0}} \hat{s_1}...\hat{s_m}$ is the abstract counterexample of DA corresponding to h, if $\hat{s_i} = \alpha(q_i, \mathbf{x}_i)$ holds $\forall i \in \{0, ..., m\}$. Reciprocally, given a counterexample $\hat{h}$ of DA, h is called a corresponding concrete counterexample if $\hat{s_i} = \alpha(q_i, \mathbf{x}_i)$ and $(q_i, \mathbf{x}_i) \xrightarrow{l_i} (q_{i+1}, \mathbf{x}_{i+1}) \in \rightarrow$, where $\rightarrow$ is a transition relation of the labeled transition system representing the semantics of H. If a counterexample $\hat{h}$ of DA has no corresponding concrete counterexample of H, then $\hat{h}$ is called a spurious counterexample. The following theorem expresses that if the safety property is satisfied at the abstract level then it is satisfied at the concrete level.

Theorem 2.1. *Given a hybrid automaton H, a discrete automaton DA which is an abstraction of H with abstraction function α, let $B \subset S$, and $\hat{B} = \{\hat{b} \mid \exists b \in B : \hat{b} = \alpha(b)\}$. Then $DA \models \boldsymbol{AG}\neg\hat{B}$ implies $H \models \boldsymbol{AG}\neg B$.*

Now we introduce a scheme for CEGAR to verify safety properties for a given hybrid automaton, whose idea is to repeat the following steps until the considered property is verified or refuted (it is assumed that $\alpha(q_i, \mathbf{x}_{i+1}) \in \hat{B}$ implies $(q_i, \mathbf{x}_{i+1}) \in B$):

1. analyze $DA \models \mathbf{AG}\neg\hat{B}$ with model checking:
    - if the property holds, return the conclusion that H is safe from Theorem 2.1;
    - if it does not hold, a counterexample is obtained before turning to 2;

2. validate whether this counterexample has a corresponding concrete counterexample in H:

    - if there does exist a corresponding concrete counterexample, then return that H is not safe;
    - otherwise, the counterexample is spurious and thus is used to refine DA before returning to 1 by replacing DA with the newly obtained refined abstraction.

2.4 System Diagnosability

In this section we review works concerned with diagnosability verification of hybrid systems and the particular classes of discrete event systems and timed automata.

Diagnosability Fault diagnosis is a crucial and challenging task in the automatic control of complex systems, whose efficiency depends on the system property called diagnosability. This is a property describing the ability to determine without ambiguity whether a fault of a given type has effectively occurred based on the observations provided by the system. Diagnosability analysis has already received considerable attention in the literature over latest decades. However, most of the existing works refer to discrete event systems [95, 32, 89, 97, 59, 107, 57] with stochastic and fuzzy variants [101, 77, 68] or continuous systems [23, 11, 87, 104]. Diagnosability was also studied in the framework of decentralized and distributed architectures [90, 93, 106, 83]. But many modern technological processes exhibit both continuous evolution and discrete transitions, whose combination is at the origin of complex behavior and important phenomena of such systems. To the best of our knowledge, very few works handle diagnosability of hybrid systems with satisfactory results. Given a model that describes both correct and faulty behavior and what is presumed observable (i.e., generated externally by the model), then the model is said to be diagnosable if the observations are enough to determine that a fault has occurred and, in the case of multiple faults, to identify which one of them happened. The diagnoser is a function which, given an observable trace of the system, is able to decide whether or not a fault has occurred. Thus, the model is diagnosable if and only if such a function exists. For example, if a and b are the only two observable events and u and f are two unobservable events where f corresponds to the fault event, then given a generated observable trace ab, one cannot determine, with certainty, whether the model behavior (i.e., combined with the unobservable events) corresponds for example to ab or aub or afb (if the three may exist) and thus whether the faulty event f occurred or not.

Diagnosability of discrete event systems As a first step, [103] proved that the existing definitions of diagnosability for discrete event systems and for con-

tinuous systems can be stated as a property of the system fault signatures (w.r.t. normal behavior signatures), and a unified definition of diagnosability was established. However hybrid systems diagnosability was not considered. Among the contributions concerned with hybrid diagnosability, we can mention [18] that slightly modified the classical necessary and sufficient condition for diagnosability of a discrete event system of [95] and expressed it in terms of reachability. [51] generalized this condition requiring more restrictive hypotheses. Despite the claim that the two above methods deal with hybrid systems, these works do not really account for the hybrid nature of the system as they use only a very high level discrete abstraction and ignore the continuous dynamics. On the other hand, in [35], diagnosability is expressed in terms of mode discernability (also called distinguishability by other authors) and is only based on the continuous dynamics.

Diagnosability of time-dependent systems (timed automata) Diagnosability for systems with time constraints classically modeled using finite state machines with clocks (i.e., real valued variables with constant dynamics) has been studied [105]. This type of models describes a more general class of systems than discrete event systems (DES), the clocks can measure the sojourn time in a state and constraints to enable a transition can be based on the clock valuations. *Timed automata*, a class of hybrid automata, are extensively used to model finite state machines with clock variables. Diagnosability of timed automata reveals a more challenging task than DES: in fact, the executions generated by discrete event systems are regular and many efficient algorithms exist for solving emptiness and reachability problems. This is possible since such models can be determinized. However, timed automata and more generally hybrid automata, cannot be usually turned to deterministic models, hence property verification becomes more complex at the expense of expressiveness. When a feasible diagnoser exists, the model is diagnosable. Some works aimed further at finding a deterministic diagnoser which itself is a timed automaton. It has been shown that for deterministic timed automata (DTA), it is decidable to check whether a diagnosis function is realizable and that its complexity class is $2EXPTIME$ [21]. This class supposes that any two outgoing transitions from a given state cannot be labeled by the same event as long as their guards are enabled for the same valuations (i.e., the events are

time-deterministic). A method describing the conditions to obtain a diagnoser as a deterministic timed automaton is also described.

Diagnosability of Hybrid Systems [14] was among the early works that coped with actual hybrid systems, introducing the idea to consider a hybrid model as a twofold mathematical object. A hybrid system is modeled as a hybrid automaton whose discrete states represent its operation modes for which the continuous dynamics are specified. The discrete event part (automaton) constrains the possible transitions among modes and is referred to as the *underlying DES*. The restriction of the hybrid system to the continuously-valued part of the model is defined as the *multimode system*. Considering the analytical redundancy approach to define a set of residuals [58] for every mode, [14] introduced the concept of *mode signature* which refines the classical concept of fault signature. Mode signatures determine mode distinguishability. The key idea of [14] is to abstract the continuous dynamics by defining a set of "diagnosis-aware" events, called *signature-events*, associated to mode signature changes across modes. Signature-events are used to enrich appropriately the underlying DES. The behavior of the abstract system is then modeled by a prefix-closed language over the alphabet enriched by these additional events. The finite state machine generating this language is called the *behavior automaton*. Based on the *abstract language*, the diagnosability analysis of the hybrid system is cast into a discrete event framework and standard methods of this field can be used.

The approach of [14] later consolidated in [16] can be compared to the approach proposed in [38, 39] which uses fault signatures to capture the continuous dynamics. The fault signatures of [38, 39] are based on fault transients and they directly express the expected dynamic behavior of measured variables after the fault abstracted in qualitative terms. The approach of [14, 16] differs in that it uses mode signatures that are specifically built for diagnosis, based on standard analytical redundancy residual methods of the FDI control field [46]. Its originality relies in that it proposes a way to integrate these methods with equally standard methods of the DES diagnosis field [112]. [14, 16] adopt the diagnoser approach [95] because it has the advantage to also support straightforwardly online diagnosis. [45] repeats these ideas differing by the fact that the diagnoser is directly built

from the underlying DES and mode distinguishability is used to cluster its state labels. This method leads to a so-called *clustered diagnoser*. Let us note that this method only applies to a restricted class of hybrid systems for which transitions triggered by continuous dynamics are not allowed.

Checking DES diagnosability with methods based on the construction of diagnosers has exponential complexity with the size of the underlying DES automaton. Hence, approaches based on *verifiers*, also known as *twin plant* approaches, are generally preferred. This is because, although a twin plant cannot be used for online diagnosis, it can be constructed in polynomial time. Methods integrating a twin plant approach with mode distinguishability checking for assessing hybrid system diagnosability are recent. The reader can refer to [44] as a first piece of work in this direction. Later, [60] indicated that mode distinguishability could be complemented by another property of the continuous dynamics named *ephemerality*. Ephemerality states when the system cannot stay forever in a given set of modes. The continuous dynamics are hence abstracted remembering only these two pieces of information. In addition to this, [60] checks diagnosability in an incremental way. It starts by generating the most abstract DES model of the hybrid system and checking diagnosability of this DES model. A "counterexample" that negates diagnosability is possibly provided based on the twin plant. The model is then refined to try to invalidate the counterexample and the procedure repeats as far as diagnosability is not proved. This approach hence uses just the necessary information about continuous dynamics, in an "on request" manner, hence making the best out of computation. In the most recent literature concerned with hybrid system diagnosability like [60] and also [42], which characterizes the maximum delay for diagnosing faults given measurement uncertainty, abstraction is key. Abstraction is also at the core of other methods to check other properties of hybrid systems.

Diagnosability via abstraction [43] applied the use of abstractions as timed automata to diagnosability verification and showed that for a certain class of abstractions, the verification problem belongs to the complexity class P. [42] presents a procedure to determine the (δ_d, δ_m)-diagnosability of a hybrid system where δ_d

is the maximum time to diagnose a fault after its occurrence and δ_m is a precision metric representing the uncertainty in measuring the observable events. The method, given a hybrid automaton H with initial set of hybrid states S_0, a precision metric ϵ, a time horizon T_{end}, operates as follows. Find a set of so-called *robust* neighborhoods around elements from S_0 where the executions initially in one of these neighborhoods are alike, having the same observable events and are consistent in time until the specified horizon T_{end}. This similarity relation is defined formally using Hausdorff distance metrics guaranteeing the time consistency up till a certain imposed precision ϵ. Continue finding neighborhoods until S_0 is fully covered by them. The abstraction is formed by the many sets of executions from each of the computed neighborhoods. Proofs of the inclusion of the timed language of the hybrid automaton into the constructed abstraction are given.

2.5 Tools and Challenges in Hybrid Systems Verification

2.5.1 Numerical simulation

Acumen testbed for cyber-physical systems [100] Acumen is an experimental modeling and simulation environment for hybrid systems. It is built around a small, textual modeling language. Different plotting libraries and visualization are embedded into graphical user interfaces, providing an environment for researchers in the simulation field to run experiments.

2.5.2 Flow-pipe tools

Ariadne [88] Ariadne is an open source library developed for hybrid automata modeling and verification. It provides structures and types for the analysis and description of hybrid automata. In addition it holds state reachability computation algorithms [17].

SpaceEx [52] is a platform for the verification of hybrid systems. SpaceEx allows the use of different types of state representation and refinement strategies and combination of flow-pipes with symbolic techniques.

Flow* [26] Flow Star (Flow*) is a reachability analysis tool for hybrid systems with non linear dynamics. It computes a flow-pipe, represented by a Taylor model, which is an over-approximation of the set of reachable states.

Cora [2] Cora is a Matlab based library for formal verification and analysis of hybrid systems. Cora embeds different state representations such as polytopes, zonotopes and Taylor models.

2.5.3 SMT solvers

DReach [72] DReach encodes a given hybrid automaton and a safety specification as SMT formulas, then uses DReal for solving the obtained set of constraints.

2.5.4 Automated proof tools

KeyMaera [92] is a theorem prover dedicated to hybrid systems verification. KeyMaera uses differential dynamic logic ($d\mathcal{L}$) to write hybrid programs [91]. These programs have precise semantics and can model non-deterministic choices. The dynamics can be expressed as non linear differential equations, inequalities and non deterministic inputs. The tool combines the use of different results from algebraic calculus. A hybrid program is defined using the following grammar:

$$hp_1, hp_2 \quad ::= \quad x := \gamma \mid ?\theta \mid x' = \gamma \& \theta \mid hp_1 \cup hp_2 \mid hp_1; hp_2 \mid hp_1^*$$

Less formally, respectively:

- assignment of an expression γ to a variable x
- test θ over the variables which acts as a guard: if $?\theta = true$ then it is possible to transit, else no transition is made
- formula for the derivative x' of x with respect to time
- non deterministic choice to go through hp_1 or hp_2
- sequence ordering of the execution, hp_1 first then hp_2
- iteration of hp_1 an arbitrary finite number of times

Forms of the expressions An expression can be: $[a := -v; x' = x + y' + a]\phi$, the brackets "[ ]" refer to the LTL operator *always*, in other terms, in all reachable states, ϕ is satisfied.

Example 2.6 (Hybrid Program)**.** Hybrid programs are an equivalent representation method of hybrid automata. To illustrate a hybrid program, let us consider the verification task of a simple train control system. The example is present in Keymaera. The goal is to verify that the train at location z never bypasses the movement authority (MA) by a distance SB in order to avoid collisions with other train. In $d\mathcal{L}$ logic, this is expressed by a hybrid program where a is the acceleration of the train, b is the train braking constant and c is an acceleration constant such that:

$$\begin{aligned} ETCS \quad \equiv \quad & if\ (MA - z < SB)\ then\ a := -b \\ & else\ a := c \\ & z'' = a \end{aligned}$$

- The $ETCS$ program consists of two parts, a control part *ctrl* (verifying a guard condition with a reset) and a continuous applied dynamic *drive*. When embedded into an actual train, the $ETCS$ program must repeat itself periodically. This is represented by assigning the Kleene star to the sequence: $ETCS \equiv (ctrl; drive)^*$. The star refers to continuous repetition of the sequence $ctrl; drive$ of hybrid programs with respect to time.
- *ctrl* evaluates the condition on the MA, if it is verified then the train acceleration can be a positive constant (acceleration mode), else it is negative (brake mode). Thus, *ctrl* decides whether a discrete jump is applied or not.
- *drive* represents the continuous evolution in the hybrid system specified by the differential equation $z'' = a$

KeyMaera takes as input the hybrid program and a formula to verify such as $\phi := z \leqslant MA$. Temporal logic is used to state that ϕ should be verified in all states of the controller. The tool then applies proof strategy over the hybrid program to deduce the truth value of ϕ w.r.t the $ETCS$ program.

2.5.5 Challenges in Hybrid Systems Verification

Cross Disciplinary The study of hybrid systems involves different disciplines such as applied mathematics, control theory, theoretical and practical computer science. Consequently, the classical education curriculum taking one of the fields as a specialty is not fit for the study of hybrid systems and cyber-physical systems in general.

Large scale systems The verification of large scale hybrid systems (i.e., with the number of coupled variables exceeding around 40) remains a challenge. The future aim of the scientific community will be to develop formalism adapted to larger systems such as in distributed framework. Many applications arise in distributed hybrid systems such as managing a fleet of unmanned vehicles or aircrafts.

Distributed Hybrid Systems Very few works tackle the verification of hybrid systems in a distributed framework. The large scale complexity can be relaxed through distribution and local computations.

Standardization There is a lack of standardized specification language for cyber-physical and hybrid systems.

2.6 Chapter Summary

The chapter introduced model based verification and reviewed the existing literature concerned with qualitative modeling, reasoning and simulation, then the existing hybrid systems verification techniques, tools and challenges and the literature concerned with the diagnosability property. In fact, qualitative reasoning allows modeling systems with partially known knowledge while still getting conclusions via simulation. We have seen that it is crucial to precisely denote the assumptions such as continuity of functions, differentiability and smoothness, as these hypotheses are a basis for computing correct successors of an initial given qualitative state. We reviewed different verification techniques for verifying hybrid systems. In particular Flow-pipe computation aims at finding over and under-approximations of

the reachable set of states of a hybrid system that are semantically correct but still not computationally expensive.

Chapter 3

Framework for Hybrid Automata Verification

3.1 Hybrid Automata

In this part we introduce the formal framework describing hybrid automata that will be adopted throughout the thesis and the different notations that will be used. Later on, we provide an example of a practical system modeled as a hybrid automaton. Lastly, we formally introduce the different classes of hybrid automata, in which timed and polynomial hybrid automata are of our primary interest.

3.1.1 Hybrid Automata overview

Hybrid systems are dynamical systems that include discrete and continuous behaviors [65]. **Hybrid automata** (HA) are a mean to model such systems. A hybrid automaton is an infinite state machine. Each state of the hybrid automaton is twofold with a discrete and a continuous part. The discrete part ranges over a finite domain while the continuous part ranges over the Euclidean space $\mathbb{R}^n$. Intuitively a hybrid automaton permits modeling continuous change and discrete behavior such as digital electronics controlling a physical environment. The main components of a hybrid automaton are:

- **Graph:** A graph of nodes and edges, where nodes are Modes (or *Locations*, or *Discrete State*) of the system and edges representing control switches,

each edge relates unidirectionally two modes with each other and expresses a mode change.

- **Continuous Dynamics (or *Flow Condition*):** Each mode of the graph, is assigned a set of constraints, describing the continuous change. More precisely, continuous dynamics can be expressed as differential equations (or inclusions) or, in a much simpler form, a set of the possible valuations of the derivative such as an interval over the reals.

- **Variables:** A finite set of containers, whose valuations are over the set of real numbers.

- **Jump Conditions:** A set of constraints assigned to each mode. When a jump constraint is satisfied, a switching between two related modes is allowed to occur.

3.1.2 Hybrid Automata Definition

We now propose a general formal definition of hybrid automata and introduce the different notations that will be used throughout this thesis.

Definition 3.1 (Hybrid Automata (HA))**.** *An n-dimensional hybrid automaton (HA) is a tuple* $H = (Q, X, S_0, \Sigma, F, Inv, T)$ *where:*

- Q *is a finite set of modes (or locations), that can be possibly defined as the valuations set of a finite number of finite valued variables, and represents the* discrete *part of* H*.* X *is a set of* n *real-valued variables (which are locally continuously differentiable functions of time), whose valuations set* $\boldsymbol{X} \subseteq \mathbb{R}^n$ *represents the* continuous *part of* H*.* $S = Q \times \boldsymbol{X}$ *is the state space of* H*, whose elements, called states, are noted* $(q, \boldsymbol{x})$ *with* q *and* $\boldsymbol{x}$ *the respective discrete and continuous parts of the state.*

- $S_0 \subseteq S$ *is the set of initial states. If unique, it is noted* $(q_0, \boldsymbol{x}_0)$*.*

- Σ *is a finite set of events.*

- *$F : S \to 2^{\mathbb{R}^n}$ is a mapping assigning to each state $(q, \boldsymbol{x}) \in S$ a set $F(q, \boldsymbol{x}) \subseteq \mathbb{R}^n$ constraining the time derivative $\dot{\boldsymbol{x}}$ of the continuous part of the mode q by $\dot{\boldsymbol{x}} \in F(q, \boldsymbol{x})$. If there is no uncertainty on the derivative, then F is a function $S \to \mathbb{R}^n$ specifying the flow condition $\dot{\boldsymbol{x}} = F(q, \boldsymbol{x})$ in each mode q (the dynamics in each mode is thus given by a set of n first-order ordinary differential equations (ODEs)).*

- *$Inv : Q \to 2^{\boldsymbol{X}}$ assigns to each mode q an invariant set $Inv(q) \subseteq \boldsymbol{X}$, which constrains the values of the continuous part of the state while the discrete part is q. We require, for all $q \in Q$, that $\{\boldsymbol{x} \mid (q, \boldsymbol{x}) \in S_0\} \subseteq Inv(q)$.*

- *$T \subseteq S \times \Sigma \times S$ is a relation capturing discontinuous state changes, i.e., instantaneous discrete transitions from one mode to another one. Precisely, $t = (q, \boldsymbol{x}, \sigma, q', \boldsymbol{x}') \in T$ represents a transition whose source and destination states are $(q, \boldsymbol{x})$ with $\boldsymbol{x} \in Inv(q)$ and $(q', \boldsymbol{x}')$ with $\boldsymbol{x}' \in Inv(q')$, respectively, and labeled by the event σ. It represents a jump from $\boldsymbol{x}$ in mode q to $\boldsymbol{x}'$ in mode q'.*

We will call *(concrete) behavior* of H any sequence of continuous solution flows and discrete jumps, rooted in an initial state, satisfying all the constraints above defining H.

Set Based Notations Hybrid systems are typically represented as finite automata with (discrete, i.e., modes) states Q, initial states $Q_0 = \{q \in Q \mid \exists \mathbf{x} \in Inv(q)\ (q, \mathbf{x}) \in S_0\}$ and transitions δ defined by $\delta = \{(q, \sigma, q') \in Q \times \Sigma \times Q \mid \exists \mathbf{x}, \mathbf{x}'\ (q, \mathbf{x}, \sigma, q', \mathbf{x}') \in T\}$. To each state $q \in Q_0$ is associated an initial (continuous) nonempty set $Init(q) = \{\mathbf{x} \in Inv(q) \mid (q, \mathbf{x}) \in S_0\}$. To each transition $\tau = (q, \sigma, q') \in \delta$ are associated a nonempty guard set $G(\tau) = \{\mathbf{x} \mid \exists \mathbf{x}'\ (q, \mathbf{x}, \sigma, q', \mathbf{x}') \in T\} \subseteq Inv(q)$ and a set-valued reset map $R(\tau) : G(\tau) \to 2^{Inv(q')}$ given by $R(\tau)(\mathbf{x}) = \{\mathbf{x}' \mid (q, \mathbf{x}, \sigma, q', \mathbf{x}') \in T\}$. It is actually equivalent in the definition to provide either T or δ, G and R. In the last case, H is denoted by $(Q, X, S_0, \Sigma, F, \delta, Inv, G, R)$ and we have: $\forall (q, \mathbf{x}), (q', \mathbf{x}') \in S$, $\forall \sigma \in \Sigma$, $((q, \mathbf{x}, \sigma, q', \mathbf{x}') \in T \Leftrightarrow \tau = (q, \sigma, q') \in \delta \wedge \mathbf{x} \in G(\tau) \wedge \mathbf{x}' \in R(\tau)(\mathbf{x}))$.

Relational Based Notations It can be in some cases more convenient to adopt a relational-based representation than a set-based representation and to use predicates instead of subsets. By a slight abuse of notation, for each mode q, $Init(q)$ (for $q \in Q_0$), $F(q)$ and $Inv(q)$ indicate then predicates whose free variables are respectively from X, $X \times \dot{X}$ and X and $Init(q)(\mathbf{x})$, $F(q)(\mathbf{x}, \dot{\mathbf{x}})$ and $Inv(q)(\mathbf{x})$ being true means respectively $\mathbf{x} \in Init(q)$, $\dot{\mathbf{x}} \in F(q, \mathbf{x})$ and $\mathbf{x} \in Inv(q)$. In the same way, for each mode transition τ, $G(\tau)$ and $R(\tau)$ indicate predicates whose free variables are respectively from X and $X \times X$ and $G(\tau)(\mathbf{x})$ and $R(\tau)(\mathbf{x}, \mathbf{x}')$ being true means respectively $\mathbf{x} \in G(\tau)$ and $\mathbf{x}' \in R(\tau)(\mathbf{x})$. We will make use equally of both representations.

Assumption 3.1. *Guards in any mode q will be assumed non-intersecting:* $\forall q \in Q, \forall \tau_1 = (q, \sigma_1, q_1) \in \delta, \forall \tau_2 = (q, \sigma_2, q_2) \in \delta, (\tau_1 \neq \tau_2 \Rightarrow G(\tau_1) \cap G(\tau_2) = \emptyset)$.

Non-deterministic state machine Thus, at any moment of its continuous evolution in a mode q, the system may jump to at most one other mode and by a unique event. Nevertheless, a HA is generally non-deterministic: the continuous dynamics in each mode may be non-deterministic, the moment where a jump occurs is non-deterministic (as long as $Inv(q)(\mathbf{x})$ and $G(\tau)(\mathbf{x})$ are true, where q is the source mode of the mode transition τ, the system may continue to continuously evolve in q or make the transition τ) and the reset after a jump may be non-deterministic.

3.1.3 Graphical representation

A hybrid automaton is visually represented by a set of nodes and arcs:

- **Nodes:** each node represents a **mode** of the system and is labeled by the flow constraints such as differential equations followed by the mode invariant sets.
- **Arcs:** each arc represents a **jump** (discrete transition) from one mode to another. The arc is labeled by its event and its associated guard (and possibly its reset map).

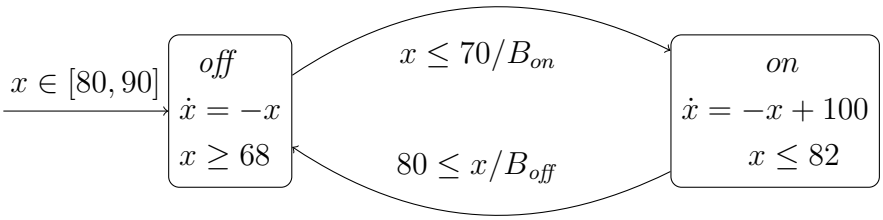


Figure 3.1: 1-dimensional hybrid automaton modeling a thermostat

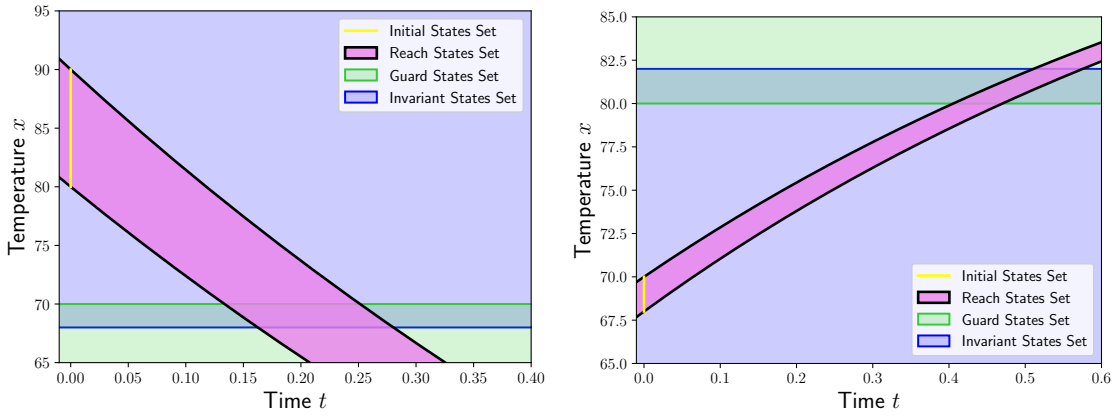
Example 3.1 (Temperature regulator)**.** A temperature regulator or thermostat system maintains the temperature of an object quasi-constant by turning on and off a heater device. Practically speaking, such system contains at least, a temperature sensor, a heater device and logic control electronic circuits. An algorithm computes, given the actual measured temperature of the object, the corresponding order for the circuitry to activate or not the heater. A hybrid automaton $H = (Q, X, S_0, \Sigma, F, \delta, Inv, G, R)$ models the behavior of such system (Figure 3.1) such that:

- $Q = \{on, off\}$, $X = \{x\}$, $S_0 = (off, [80, 90])$
- $\Sigma = \{B_{on}, B_{off}\}$, $F(on) = \{\dot{x} = -x + 100\}$, $F(off) = \{\dot{x} = -x\}$
- $\delta = \{\tau_1 = (off, B_{on}, on), \tau_2 = (on, B_{off}, off)\}$, $Inv(off) = x \geq 68$, $Inv(on) = x \leq 82$
- $G(\tau_1) = x \leq 70$, $G(\tau_2) = x \geq 80$, $R(\tau_1) = R(\tau_2) = (x = x')$

The assigned hybrid automaton H is one dimensional, where x represents the sensed temperature.

3.1.4 Hybrid Automata semantics

The semantics of a hybrid automaton (i.e., the set of executions of a HA) is a set of sequences, every sequence aggregates continuous time lapses with mode jumps while satisfying all the flow, invariant and jump constraints. We denote by $q_0, q_1, ...$ the modes of Q, and by $x_1, x_2, ..., x_n$ the variables of X.



(a) (b)

Figure 3.2: Thermostat behavior for: (a) Mode *off*, (b) Mode *on*

Definition 3.2 (Hybrid automaton semantics)**.** *The semantics of a hybrid automaton H, denoted by $[[H]]$, is the set of all executions (or runs), which are labeled sequences of states from S with labels in $L = \Sigma \cup \mathbb{R}_+$: $(q_0, \boldsymbol{x}_0) \xrightarrow{l_0} (q_1, \boldsymbol{x}_1) \ldots (q_i, \boldsymbol{x}_i) \xrightarrow{l_i} \ldots$ such that $(q_0, \boldsymbol{x}_0) \in S_0$ and, for any two successive states $(q_i, \boldsymbol{x}_i) \xrightarrow{l_i} (q_{i+1}, \boldsymbol{x}_{i+1})$ in the sequence, one of the following is true:*

- $l_i = \sigma_i \in \Sigma$ *and* $(q_i, \boldsymbol{x}_i, \sigma_i, q_{i+1}, \boldsymbol{x}_{i+1}) \in T$*;*
- $l_i = d_i \in \mathbb{R}_+$, $q_i = q_{i+1}$, $\boldsymbol{x}_i, \boldsymbol{x}_{i+1} \in Inv(q_i)$ *and* $\exists x : [0, d_i] \to \boldsymbol{X}$ *continuously differentiable function, with* $x(0) = \boldsymbol{x}_i$, $x(d_i) = \boldsymbol{x}_{i+1}$ *and* $\forall t \in (0, d_i)$ $\dot{x}(t) \in F(q_i, x(t))$ *and* $x(t) \in Inv(q_i)$*.*

In the first case, the system executes a discrete transition (also called discrete jump) $\tau_i = (q_i, \sigma_i, q_{i+1})$ from the source mode q_i to the destination mode q_{i+1}. Such a transition is possible (enabled) as soon and as long as $\mathbf{x}_i \in G(\tau_i)$. After the jump, the system may follow the new dynamics given by $F(q_{i+1})$, starting from the continuous state $\mathbf{x}_{i+1} \in R(\tau_i)(\mathbf{x}_i)$. Notice that no time elapses during a discrete jump, which is instantaneous. In the second case, the system performs a continuous transition (also called continuous flow) of duration d_i inside the mode q_i, constrained by the dynamics $F(q_i)$ and the invariant set $Inv(q_i)$. The sequence $h = (off, 80) \xrightarrow{0.15} (off, 69) \xrightarrow{B_{on}} (on, 69) \xrightarrow{0.5} (on, 81) \xrightarrow{B_{off}} (off, 81) \ldots$ is valid for the thermostat (Example 3.1), thus $h \in [[H]]$. The trace of an execution h, i.e.,

the sequence of its labels, is a word from $L^\star$ (or L^ω for infinite h), denoted as $trace(h)$. We denote the total time duration of h by $time(h) \in \mathbb{R}_+ \cup \{+\infty\}$, which is calculated as the sum of all time periods in the trace of h: $time(h) = \sum d_i$. Figure (3.2) illustrates the behavior of the thermostat system: the reachable set of states starting from the initial set S_0, the guard and invariant states sets.

Zeno behavior An execution $h \in [[H]]$ is zeno if it is infinite with finite time, i.e., the sum given by

$$\sum_{i=0}^{+\infty} d_i$$

converges (i.e., tends to a real finite value). Consequently, in a zeno execution h, an unbounded number of discrete jumps occur while the execution time $time(h)$ is finite. Real systems in general do not show zeno behavior. Such behavior arises at the modeling stage and is due to the abstraction and incompleteness of the model. Many works addressed the conditions for zeno behaviors to exist [113].

Assumption 3.2. *We limit our work to cases that do not account for any zeno behavior.*

Example 3.2 (Bouncing Ball)**.** The classical bouncing ball example illustrates zeno behavior. A ball is initially released from some height H_0 without initial speed. Once it hits the ground, the ball bounces back. The example at some abstraction level, can be modeled by a hybrid automaton with a single mode and one jump. The variables of the system are: the vertical position of the ball x_0 and its velocity $x_1 = \frac{dx_0}{dt}$. The forces applied to the ball are limited to the gravitational force, thus $\frac{dx_1}{dt} = -g$ is constant with g the gravitational acceleration of the Earth. The ball is assumed semi-elastic and hollow, so when the collision with the ground occurs some of the air inside the ball is compressed for a short period of time due to the deformation of the ball. The ball is well sealed, so while the ball takes back its normal form, the compressed air allows the ball to bounce again. Due to friction with the ground, some of the ball's kinetic energy is lost and some is restituted. The energy loss and restitution happen in a short duration of time when compared with the time elapsed between two successive bounces of the ball. Consequently, the energy loss can be assumed instantaneous, the change of speed is modeled as

a discrete jump back to the same mode while having a different value of the speed (because of the energy loss, modeled as a reset) and an opposite direction of the velocity after the jump (because the ball does not penetrate through the ground). The guard holds the condition $x_1 < 0 \wedge x_0 = 0$ and the associated reset, $x_1 := -r.x_1$ with $r \in (0, 1)$ the restitution factor. Using the reachability computation tool *CORA* initialized from the set $[0.9, 1.1]$ we compute the flow-pipe till a time horizon of 2.1 time units using zonotopes as state representation (Figure 3.3). We can clearly see that the bouncing ball simulated trajectory exhibits zeno behavior: in a bounded area of x_0 such as $[0, 0.2]$ the number of bounces increases infinitely often.

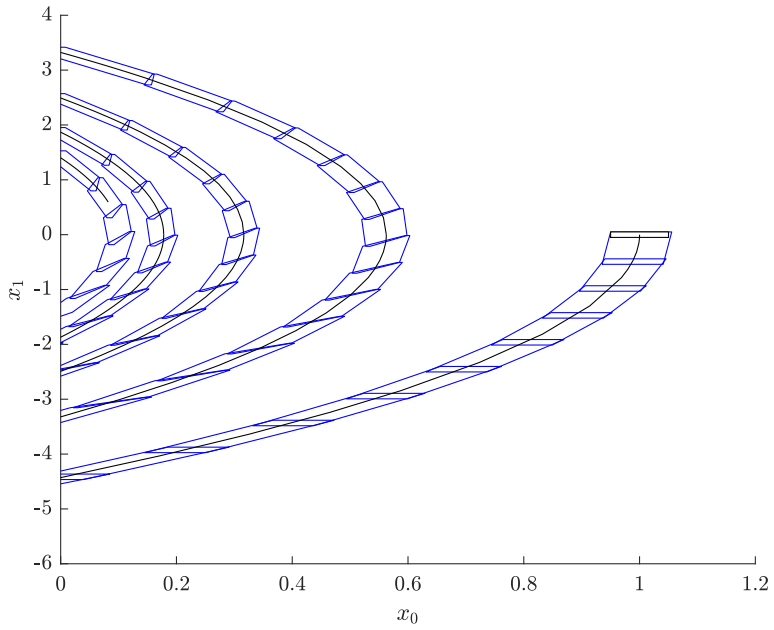


Figure 3.3: Flow-pipe of the bouncing ball simulated using zonotopes

3.1.5 Hybrid Automata Classes and Particular Cases

The components of the hybrid automaton previously defined can be further constrained to find particular classes. On the two opposite sides of the continuity spectrum we find discrete automata and continuous systems. When constraining primarily the dynamics F we find practical classes of hybrid automata such as

the rectangular or linear class or timed automata. For the rectangular class the dynamics valuations are a cartesian product of intervals and it lies on the boundary of the decidability over reachability problem with some restrictions [66]. The timed and polynomial hybrid automata classes are of particular interest for this thesis work.

Definition 3.3 (Discrete Automata (DA))**.** *It is the case where there is no continuous space. Thus, a (finite) discrete automaton (DA) is a tuple* $D = (Q, Q_0, \Sigma, \delta)$ *where:*

- Q *is a finite set of discrete states (modes).*
- $Q_0 \subseteq Q$ *is the set of initial states.*
- Σ *is a finite set of events.*
- $\delta \subseteq Q \times \Sigma \times Q$ *is a set of transitions of the form* $\tau = (q, \sigma, q')$*.*

This definition is identical to Definition 2.6, except that now any state is considered final. The semantics $[[D]]$ of D is given by the set of sequences (called paths) made up of successive states transitions labeled by events and rooted in an initial state. It has a finite representation. The trace of such a path is the word in $\Sigma^\star$ (Σ^ω for infinite paths) whose letters are the successive labels of the path.

Definition 3.4 (Continuous systems (CS))**.** *It is the case where there is no discrete part. Thus, an* n*-dimensional continuous system (CS) is a particular hybrid automaton* C *with only one mode (*$|Q| = 1$*) and* $\Sigma, T = \emptyset$ *(and thus* δ, G, R *too). It can thus be denoted as* $C = (X, S_0, F, Inv)$ *with* $S_0 \subseteq Inv$*.*

The semantics $[[C]]$ of C is the set of all time labeled sequences of continuous states, rooted in an initial state, corresponding to the continuous transitions constrained by the dynamics F and the invariant set Inv. Its representation is infinite.

Timed Automata It is a class of hybrid automata where the continuous variables x_i, $1 \leq i \leq n$, with values in $\mathbb{R}_+$, called clocks, have all first order derivatives equal to one. So time elapses identically for all clocks. The set $C(X)$ of constraints over a set of clocks X is defined as follows: a constraint is either a primitive constraint of the form x_i *op* c_i where $c_i \in \mathbb{R}_+$ (at the theoretical level because, in practice, $\mathbb{Q}$ is used instead of $\mathbb{R}$ for computer implementation reasons) and *op* is one of $<, \leq, =, \geq, >$ or a finite conjunction of primitive constraints. The satisfiability set of a constraint is thus a rectangle in $\mathbb{R}_+^n$, i.e., the product of n intervals of the half real line $\mathbb{R}_+$, and we will identify $C(X)$ to the set of rectangles.

Definition 3.5 (Timed automata (TA))**.** *A timed automaton (TA) is a hybrid automaton* $T = (Q, X, S_0, \Sigma, F, \delta, Inv, G, R)$ *such that:*

- $\boldsymbol{X} = \mathbb{R}_+^n$.
- $S_0 = Q_0 \times \{\boldsymbol{0}\}$.
- $\forall q \in Q \ F(q, .) = \boldsymbol{1}$*, which means that the dynamics of clocks evolution in each mode* q *is given by* $\dot{x}_i = 1$.
- $Inv : Q \to C(X)$ *associates to each mode* q *a rectangle invariant in* $\boldsymbol{X}$*. We require* $\boldsymbol{0} \in Inv(q_0)$.
- $G : \delta \to C(X)$ *associates to each discrete transition* (q, σ, q') *a rectangle guard in* $Inv(q)$.
- $\forall \tau \in \delta \ \exists Y(\tau) \subseteq X \ \forall \boldsymbol{x} \in G(\tau) \ R(\tau)(\boldsymbol{x}) = \{\boldsymbol{x}'\}$ *with* $\boldsymbol{x}'_i = 0$ *if* $x_i \in Y(\tau)$ *and* $\boldsymbol{x}'_i = \boldsymbol{x}_i$ *otherwise, i.e., clocks in* $Y(\tau)$ *are reset to zero with transition* τ*, the others keeping their values.*

The notation of a timed automaton T is generally simplified as $T = (Q, X, Q_0, \Sigma, Inv, (\delta, G, Y))$. The semantics of T as a hybrid automaton, given by Definition 3.2, can be simplified by merging together in an execution successive timed transitions between two discrete transitions and summing up their time period labels. An execution in $[[T]]$ is thus a sequence h of alternating time steps (possibly with 0 time period) and discrete steps of the form $(q_0, \mathbf{x}_0) \xrightarrow{d_1} (q_0, \mathbf{x}_0 + d_1) \xrightarrow{\sigma_1} (q_1, \mathbf{x}_1) \xrightarrow{d_2} ...$ whose trace $trace(h)$ is the timed word $d_1\sigma_1 d_2 ... \in \mathbb{R}_+(\Sigma\mathbb{R}_+)^*$ and duration is

$time(h) = \sum d_i$.

Decidability results The class of timed automata is particularly interesting as the reachability and language emptiness problems are decidable for that class and are PSPACE-complete [7]. This result still holds for **singular automata** where the flow $F(.,.)$ is a constant singleton, allowing thus different constant slopes to the clocks. However, if a clock can switch from active (turned on) to inactive (turned off), or vice versa, when transiting between two modes then it is called a **stopwatch**, i.e., $\forall q \in Q \ \exists \mathbf{c} \in \{0,1\}^n \ F(q,.) = \mathbf{c}$, which means that the dynamics of clocks evolution in each mode q is given by $\dot{x}_i = 1$ for those clocks active in q and $\dot{x}_i = 0$ for those clocks inactive in q. During inactivity, it holds its last valuation when it was active (or 0 in case of reset). It has been shown that decidability for timed automata with one stopwatch holds if some strong conditions are met [20], however it is no longer true for timed automata with three stopwatches (and thus also for the more general class of **multisingular automata**, where the flow singletons depend on the mode).

Rectangular Automata For this class, the unique flow condition $F(.,.)$ is the same for all modes and is given by a bounded rectangle in $\mathbb{R}^n$ (instead of the singleton $\mathbf{1}$), $Init(q_0)$ is a bounded rectangle, $Inv(q)$ is a rectangle for any mode q and, for any discrete transition τ, the guard $G(\tau)$ is a rectangle and the reset $R(\tau)$ is a bounded rectangle for those reset variables, which depend only on τ [65]. The idea behind a rectangular automaton is that changes of the discrete state has no effect over the variations of the continuous variables. The variables values can change between the discrete states but not their flow. Rectangular automata naturally express that the continuous variables are decoupled.

Definition 3.6 (Rectangle of dimension n)**.** *A rectangle I of dimension n is a product of n intervals $I_i = (a_i, b_i)$ whose endpoints (that may belong or not to the interval) are in $\{\mathbb{R} \cup \pm\infty\}$*

$$I = \Pi_{1 \leqslant i \leqslant n} I_i \tag{3.1}$$

A rectangle is bounded if each of its intervals I_i is bounded.

Definition 3.7 (Rectangular Automata (RA))**.** *A rectangular automaton (RA) is a hybrid automaton* $R = (Q, X, S_0, \Sigma, F, \delta, Inv, G, R)$ *such that:*

- $\boldsymbol{X} = \mathbb{R}^n_+$.
- $\exists I, S_0 = Q_0 \times I$ *where* I *is an* n*-dimensional bounded rectangle.*
- $\exists I^F \forall q \in Q, F(q, .) = I^F$ *and* I^F *is an* n*-dimensional bounded rectangle.*
- $Inv : Q \to C(X)$ *associates to each mode* q *a rectangle invariant in* $\boldsymbol{X}$*.*
- $G : \delta \to C(X)$ *associates to each discrete transition* (q, σ, q') *a rectangle guard in* $Inv(q)$*.*
- $\forall \tau \in \delta\ \exists Y(\tau) \subseteq X\ \exists I_i(\tau)$*, for* $x_i \in Y(\tau)$*, bounded intervals* $\forall \boldsymbol{x} \in G(\tau)$ $R(\tau)(\boldsymbol{x}) = \{\boldsymbol{x}'\}$ *with* $\boldsymbol{x}'_i \in I_i(\tau)$ *if* $x_i \in Y(\tau)$ *and* $\boldsymbol{x}'_i = \boldsymbol{x}_i$ *otherwise, i.e., clocks in* $Y(\tau)$ *are reset to intervals* $I_i(\tau)$ *with transition* τ*, the others keeping their values.*

Unknown constant An unknown constant can be modeled by a variable x_j with $I^F_j = [0, 0]$. x_j should not be reinitialized by any control switch and other variables may depend on the value of x_j.

Decidability results Decidability for the reachability problem and for language emptiness problem holds for the class of rectangular automata. And thus holds for the subclass of singular automata, whose timed automata are a particular case. But this is no longer true for the larger class of **multirectangular automata** where the flow rectangle conditions $F(q, .) = I^F_q$ depend on the mode q (as written above it is already not true for the subclass of multisingular automata with singleton flows). Notice nevertheless that, allowing changes of flow conditions with changes of modes may remain manageable if, e.g., we require a reset of the variables concerned when it occurs. That is how **initialized multirectangular automata**, i.e., where for each discrete jump, each variable whose flow interval is changed in this jump has to be reset (reinitialized), can be translated to rectangular automata. Decidability does not hold any more also for the class of **triangular automata** which generalize rectangular automata by replacing rectangles by so-called triangles obtained by intersecting rectangles with any

number of half-spaces defined by inequalities of the form $x_i \leq x_j$, i.e., if variables are not pairwise independent (extending for timed automata the set $C(X)$ by constraints of the form $x_i - x_j$ *op* c_{ij} leads already to undecidability). **Linear automata** generalize both multirectangular and triangular automata by allowing sets $F(q,.), Init(q), Inv(q), G(\tau), R(\tau)(.)$ to be any convex polyhedra in $\mathbb{R}^n$ (instead of just rectangles or triangles) and different flows conditions for different modes. And **polynomial automata** generalize linear automata by allowing those sets to be defined no longer by just linear constraints but by polynomial constraints.

Time elapsed in a state In a multirectangular automaton and for a given state q, we can define a variable x_1 with flow condition in q given by ${I^F_q}_1 = [1,1]$ and in any state q' different from q given by ${I^F_{q'}}_1 = [0,0]$. It models a stopwatch and can be used to measure the duration the system stays in state q, the stopwatch being off in any other state.

Linear Hybrid Automata

A linear term over a set $Y = \{y_i, 1 \leq i \leq n\}$ of real-valued variables is a linear combination over Y with real (in practice, rational) coefficients, i.e., an expression of the form $k_0 + k_1 y_1 + \ldots + k_n y_n$ with $k_i \in \mathbb{Q}$. A linear predicate over Y is an inequality between linear terms over Y, i.e., can be written as $k_0 + k_1 y_1 + \ldots + k_n y_n \leq 0$. A convex linear predicate over Y is a finite conjunction of linear predicates over Y, thus its satisfiability set is a convex polyhedron in $\mathbb{R}^n$.

Definition 3.8 (Linear Hybrid Automata (LA))**.** *A linear hybrid automaton (LA) is a hybrid automaton* $L = (Q, X, S_0, \Sigma, F, \delta, Inv, G, R)$ *such that :*

- $\forall q \in Q, F(q)$ *is a convex linear predicate with free variables from* $\dot{\boldsymbol{X}}$ *only, which means that, in each mode,* $\dot{\boldsymbol{x}}$ *is constrained to belong to a given convex polyhedron, depending only on the mode.*
- $\forall q \in Q$, $Init(q)$ *and* $Inv(q)$ *are convex linear predicates over* $\boldsymbol{X}$*.*
- $\forall \tau \in \delta$, $G(\tau)$ *and* $R(\tau)(.)$ *are convex linear predicates over* $\boldsymbol{X}$*.*

Remark 3.1. *A linear differential equation cannot necessarily be converted into a linear flow condition: the notions of* linear *for differential equations and hybrid automata are different.*

Multi-Affine Hybrid Automata

We recall the definition of a multi-affine function [70].

Definition 3.9 (Multi-affine Function)**.** *A multi-affine function $f : \mathbb{R}^n \to \mathbb{R}^p$ is a polynomial in the indeterminates $x_1, , ..., x_n$ with the property that the degree of f in any of the variables is less than or equal to 1. Stated differently, f has the form:*

$$f(x_1, ..., x_n) = \sum_{i_1,...,i_n \in \{0,1\}} c_{i_1,...,i_n} x_1^{i_1} ... x_n^{i_n}, \tag{3.2}$$

with $c_{i_1,...,i_n} \in \mathbb{R}^p$ for all $i_1, ..., i_n \in \{0, 1\}$ and using the convention that if $i_k = 0$, then $x_k^{i_k} = 1$.

E.g., a two-dimensional multi-affine continuous system can be written as:

$$\dot{x} = ax + by + cxy + d \tag{3.3}$$

$$\dot{y} = a'x + b'y + c'xy + d' \tag{3.4}$$

where $a, b, c, d, a', b', c', d' \in \mathbb{R}$.

Definition 3.10 (Multi-Affine Hybrid Automata (MAA))**.** *A multi-affine hybrid automaton (MAA) is a hybrid automaton $MA = (Q, X, S_0, \Sigma, F, \delta, Inv, G, R)$ for which $\forall q \in Q$ the flow condition $F(q, .) : \boldsymbol{X} \to \mathbb{R}^n$ is a multi-affine function and $\forall q \in Q$, $\forall \tau \in \delta$, $Init(q), Inv(q), G(\tau), R(\tau)(.)$ are rectangle predicates.*

Polynomial Hybrid Automata

Definition 3.11 (Polynomial Hybrid Automata (PA))**.** *A polynomial hybrid automaton (PA) is a hybrid automaton $P = (Q, X, S_0, \Sigma, F, \delta, Inv, G, R)$ for which $\forall q \in Q$ the flow condition $F(q, .) : \boldsymbol{X} \to \mathbb{R}^n$ is a polynomial function and $\forall q \in Q$, $\forall \tau \in \delta$, $Init(q), Inv(q), G(\tau), R(\tau)(.)$ are defined by polynomial constraints.*

3.1.6 Modeling with hybrid automata

Hybrid automata represent an intuitive modeling framework. They are used in various domains to model complex hybrid systems. Here is a practical case where an hybrid automaton is used for modeling a system.

Example 3.3 (The switched server [81])**.** We consider an example of 4 buffers $B_1, ..., B_4$ and a server S . To each buffer B_i we assign a workflow w_i representing the rate at which work is entering the buffer. The server processes work, one buffer at a time, at a rate w_s. The switching between buffers is controlled by a user input. For the work processing to eventually terminate it is necessary that $w_s > w_1 + w_2 + w_3 + w_4$. Such system is modeled by a hybrid automaton $H = (Q, X, S_0, \Sigma, F, Inv, T)$. We first identify the discrete modes and the continuous variables. With some observation, we notice that each time the server switches buffers, the continuous change of the work waiting to be processed follows a new evolution. Therefore, a simple choice for modeling the set of discrete modes Q is that each q_i represents the server processing work at buffer B_i. Hence, we have four modes and $Q = \{q_1, q_2, q_3, q_4\}$. The server is able to switch from and to any mode, as a result, all distinct modes q_i and q_j are related by a discrete jump with a given label, say $switch\_i\_j$. Let $x_i(t)$ be the amount of work in the buffer B_i at time t. The set X of continuous variables is thus made up of those four variables x_i and $\mathbf{X} \subseteq \mathbb{R}^4$. Any state $(q_i, \mathbf{x})$ with $x_i > 0$ can be chosen as initial state: $Init(q_i)(\mathbf{x}) = (x_i > 0)$. If the system is in the discrete mode q_i, then we have $\dot{x_i}(t) = w_i - w_s$ and, for all j such that $i \neq j$, $\dot{x_j}(t) = w_j$, hence F is specified. Invariant sets are given by: $Inv(q_i)(\mathbf{x}) = (x_i \geq 0)$ and reset predicates $R(\tau)$ are the equality predicate. What is left to be specified are the values of the guards G_{ij} which in practice represent the switching protocol of the server. Finally $T = \{(q_i, \mathbf{x}, switch\_i\_j, q_j, \mathbf{x}) \mid i \neq j, \mathbf{x} \geq 0, G_{ij}(\mathbf{x}), x_j > 0\}$.

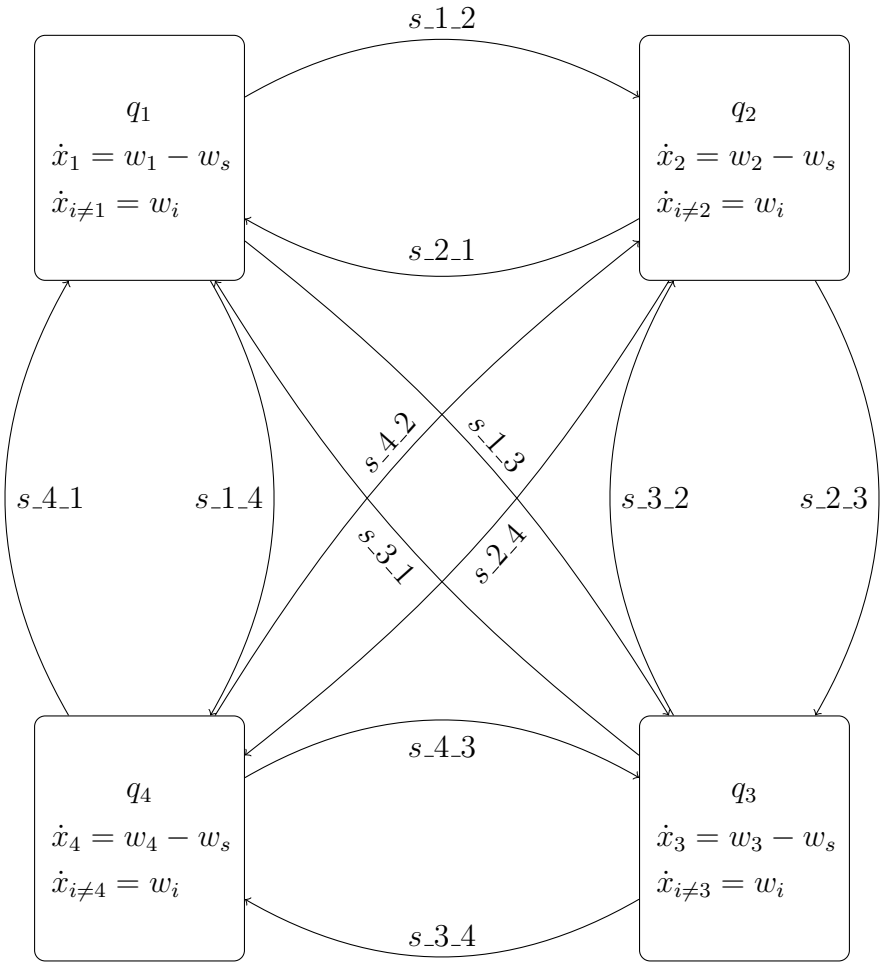


Figure 3.4: Graphical representation of the switched buffer hybrid system example

3.2 Diagnosability: Observations and Faults

In this part we introduce a formal framework for analyzing the diagnosability of a hybrid system. We specify observations and partially observable hybrid automata, we introduce fault modeling and definitions for (bounded-)diagnosability and review the important notion of critical pair.

3.2.1 Observations and faults

Remind that diagnosability is a system property allowing one to determine with certainty, at the design stage, that a fault occurred in the system, using available observations. Precisely, in a given system model, the existence of two infinite behaviors, with the same observations but exactly one containing the considered fault, violates diagnosability. Hence, to be able to analyze such property, it is necessary to define what can be observed for given systems as well as what are considered as faults. In practice, the observations are partial, only parts of the system are known and are usually obtained from sensors. In whole generality we will consider that both some discrete jumps between modes and some continuous variables inside a mode may be observable. The sets of observable events and variables are assumed to be time invariant, the second one being also assumed to be independent of the mode for the sake of simplicity. Events are observed together with their instantaneous occurrence time and variables values are assumed to be observed at any moment. E.g, for the thermostat system (Figure 3.1, page 56), transitions B_{on}, B_{off} and temperature x are assumed to be observable. For what concerns faults, we will suppose that they are modeled by some unobservable discrete jump, between precisely a normal mode and a faulty mode, translating often in a change of dynamics. This is well adapted for abrupt faults but progressive faults or degraded modes (as a shift of parameter) can be also represented in this way, the designer abstracting a slow evolution in a sudden change when he estimates that the behavior variation induced (that he will model by means of the invariant and the guard) cannot any more let consider the given mode as normal. To sum up, we obtain the following definition.

Definition 3.12 (Partially observable hybrid automaton (POHA))**.** *A partially observable hybrid automaton (POHA) is a hybrid automaton H (Def. 3.1, page. 53) where:*

- $\Sigma = \Sigma_o \uplus \Sigma_u \uplus \Sigma_f$*, i.e., the set of events is the disjoint union of the set* Σ_o *of observable (normal) events, the set* Σ_u *of unobservable normal events and the set* Σ_f *of unobservable fault events.*
- $X = X_o \uplus X_u$*, i.e., the set of continuous real-valued variables is the disjoint union of the set* X_o *of observable variables and the set* X_u *of unobservable variables.*

Definition 3.13 (Execution (timed) observation)**.** *Given an execution* $h \in [[H]]$ *of a POHA* H*,* $h = (q_0, \boldsymbol{x}_0) \xrightarrow{l_0} (q_1, \boldsymbol{x}_1)...(q_i, \boldsymbol{x}_i) \xrightarrow{l_i} ...$*, with* $l_i \in \Sigma \cup \mathbb{R}_+$*, the (timed) observation of* h *is defined as* $Obs(h) = \boldsymbol{x}_0^o, l_0^o, \boldsymbol{x}_1^o ... \boldsymbol{x}_i^o, l_i^o, ...$*, where:*

- $\boldsymbol{x}_i^o$ *is obtained by projecting* $\boldsymbol{x}_i$ *on variables in* X_o*.*
- $l_i^o = l_i$ *if* $l_i \in \Sigma_o \cup \mathbb{R}_+$*. Otherwise,* $l_i^o = \varepsilon$*, which is then removed from* $Obs(h)$*.*

Note that all durations labels $l_i = d_i$ in h are present in $Obs(h)$. Thus, any observable event $l_i = \sigma_i$ in h is present in $Obs(h)$ together with its occurrence time, obtained by adding up all durations d_j in $Obs(h)$ from the origin up to the event σ_i. In the same way, any observable variable x has its value known in $Obs(h)$ at all those instants t obtained as the sums of consecutive durations in $Obs(h)$ from the origin. If t is the occurrence time of an (observable or unobservable) event σ and if x is reset by the discrete transition σ, then the value of x changes instantaneously after this transition and the new value will be noted $x^+(t)$ to distinguish it from the value $x(t)$ before the transition (a reset observable variable may thus identify the presence of an unobservable event). Similarly, one can define observation for timed automata. The difference is that we do not assume any information about continuous clocks, so there is no $\mathbf{x}_i^o$. Then, the observation is obtained from the trace (a timed word) by erasing all unobservable events and by adding up the periods between any two successive observable events in the resulting sequence. We have thus defined what is the observation of a POHA H at the level of its timed transition system (see subsection 4.1.3). Defining its observation at the level of its

timeless transition system (see subsection 4.1.3) is similar, with $l_i \in \Sigma \cup \{\epsilon\}$ and $l_i^o = l_i$ if $l_i \in \Sigma_o$ and removed otherwise. This means that the timeless observation is obtained from the timed observation $Obs(h)$ above by removing all durations d_i, keeping thus only observable events in Σ_o and values $\mathbf{x}_i^o$ of observable variables at each transition step as an ordered sequence without any occurrence time attached.

3.2.2 System Diagnosability Definition

As we just explained, a fault is modeled as a fault event that alters the system from a normal mode to an abnormal mode. There may exist different fault events in a given system.

Assumption 3.3. *For the sake of reducing complexity (from exponential to linear in the number of different fault events) and of simplicity, in the following only one fault type, i.e., fault event, at a time is considered but multiple occurrences of this event are allowed, and the other types of fault events are thus processed as unobservable normal events.*

By processing like this successively each fault type individually, one obtains the same result about system's diagnosability as if all fault types were considered simultaneously with a lower complexity (the system is diagnosable if and only if it is diagnosable for each fault type). Now we adapt to hybrid systems the diagnosability definition [95] introduced for discrete event systems (the bounded one and the unbounded one in terms of executions lengths). h^F denotes a finite execution whose last label is a first occurrence of the fault event F considered. Given a finite execution $h \in [[H]]$ such that $h = (q_0, \mathbf{x}_0) \xrightarrow{l_0} (q_1, \mathbf{x}_1)...(q_i, \mathbf{x}_i)$, the set of post-executions of h in $[[H]]$ is defined as $[[H]]/h = \{h' = (q_i, \mathbf{x}_i) \xrightarrow{l_i} ... \mid h.h' \in [[H]]\}$, where $h.h'$ is obtained by merging the final state of h and the first state of h' (both are identical). Consequently, a faulty execution is defined as follows, where we abbreviate $F \in trace(h)$ by $F \in h$.

Definition 3.14 ((Δ-)faulty executions)**.** *Given a hybrid automaton H and F a fault event, a faulty execution is an execution $h \in [[H]]$ such that $F \in h$. Thus $h = h^F h'$ where h^F is the prefix of h whose last label is the first occurrence of F. We denote the period from (the first occurrence of) fault F in h by $time(h, F)$*

= time(h′). Given a positive real number $\Delta \in \mathbb{R}_+^*$*, we say that at least* Δ *time units pass after the first occurrence of* F *in* h*, or, in short, that* h *is* Δ*-faulty, if* $time(h, F) \geq \Delta$*.*

Definition 3.15 (Hybrid automaton (time bounded and unbounded) diagnosability)**.** *Given* $\Delta \in \mathbb{R}_+^*$*, a fault* F *is said* Δ*-diagnosable in a POHA* H *iff (if and only if)*

$$\forall h \in [[H]]\ (h\ \Delta\text{-}faulty \Rightarrow$$
$$\forall h' \in [[H]]\ (Obs(h') = Obs(h) \Rightarrow F \in h')).$$

i.e.,

$$\forall h^F \in [[H]]\ \forall h \in [[H]]/h^F\ (time(h) \geq \Delta \Rightarrow$$
$$\forall h' \in [[H]]\ (Obs(h') = Obs(h^F.h) \Rightarrow F \in h')).$$

A fault F *is said diagnosable in* H *iff*

$$\exists \Delta \in \mathbb{R}_+^*\ (F\ \Delta\text{-}diagnosable\ in\ H).$$

This definition states that F is Δ-diagnosable (resp., diagnosable) iff, for each execution h^F in $[[H]]$, for each post-execution h of h^F with time at least Δ (resp., with enough long time, depending only on F), then every execution in $[[H]]$ that is observably equivalent to $h^F.h$ should contain F. Precisely, the existence of two indistinguishable behaviors, i.e., executions holding the same observations, with exactly one containing F and time long enough after F, i.e., whose time after F is at least Δ (resp., is arbitrarily long), violates the Δ-diagnosability (resp., diagnosability) property for hybrid automata. Inspired from the framework of discrete event systems, we define critical pairs for partially observable hybrid automata taking into account both continuous and discrete dynamics.

Definition 3.16 (Δ-critical pair)**.** *A pair of executions* $h,\ h' \in [[H]]$ *is called a* Δ*-critical pair with respect to* F *iff:* $F \in h$ *and* $F \notin h'$ *and* $Obs(h) = Obs(h')$ *and* $time(h, F) \geq \Delta$*.*

We are now ready to state the sufficient and necessary condition for diagnosability verification.

Proposition 3.1. *A fault F is Δ-diagnosable in a POHA H iff there is no Δ-critical pair in $[[H]]$ with respect to F. F is diagnosable in H iff, for some Δ, there is no Δ-critical pair in $[[H]]$ with respect to F (i.e., there is no arbitrarily long time after F critical pair).*

Proof. The equivalence between the absence of Δ-critical pairs in a POHA and its Δ-diagnosability is directly seen by negating the diagnosability definition (Def.3.15):

$$\neg(\forall h \in [[H]]\ (h\ \Delta\text{-faulty} \Rightarrow \forall h' \in [[H]]\ (Obs(h') = Obs(h) \Rightarrow F \in h'))).$$
$$\Longleftrightarrow$$
$$\exists h \in [[H]]\ (h\ \Delta\text{-faulty} \wedge (\exists h' \in [[H]]\ (Obs(h') = Obs(h) \wedge F \notin h'))).$$

From the definition of a Δ-faulty execution, the previous is equivalent to

$$\exists h \in [[H]]\ (F \in h \wedge time(h, F) \geq \Delta) \wedge (\exists h' \in [[H]]\ (Obs(h') = Obs(h) \wedge F \notin h'))).$$
$$\Longleftrightarrow$$
$$\exists h, h' \in [[H]]\ (F \in h \wedge F \notin h' \wedge time(h, F) \geq \Delta \wedge Obs(h') = Obs(h)).$$

□

Note that all above definitions (e.g., observable projection, post-executions, diagnosability, critical pairs, etc.) are applicable in a similar way to timed automata, which can be considered as a special type of hybrid automata. The only difference is that the set of continuous variables is the set of clock variables whose derivative is always 1 [36, 15, 13, 37, 60]. And, as for automata the existence of arbitrarily long (in terms of transitions number) after F faulty executions implies the existence of an infinite faulty execution, in the same way it has been proved [105] that for timed automata the existence of arbitrarily long time after F faulty executions implies the existence of a $+\infty$-faulty execution (extending Definition 3.14 to $\triangle = +\infty$) and thus that non-diagnosability is witnessed by the existence of a $+\infty$-critical pair (extending Definition 3.16 to $\triangle = +\infty$) and its checking is PSPACE-complete.

Proposition 3.2. *A fault F is diagnosable in a partially observable timed automaton T iff there is no $+\infty$-critical pair in $[[T]]$ with respect to F.*

Theorem 3.1 (Diagnosability check of timed automata)**.** *Checking diagnosability of a partially observable timed automaton T is PSPACE-complete.*

Proof. The result and proof are elaborated in [105]. □

We will rest on this result as diagnosability checking of a POHA H will be performed on a timed automaton T abstracting H. We will use also the following obvious result.

Proposition 3.3. *If a POHA H is Δ-diagnosable then it is Δ^+-diagnosable for any $\Delta^+ > \Delta$. Alternatively, if H is not Δ-diagnosable then it is not Δ^--diagnosable for any positive Δ^- such that $\Delta^- < \Delta$.*

3.3 Chapter Summary

This chapter introduced a framework for hybrid automata model and semantics. We reminded classes of interest of hybrid automata. Two practical examples of modeling using the hybrid automata formalism were presented: a thermostat and a switched buffer system. The diagnosability definitions for a hybrid automaton have been presented together with critical pairs that witness non-diagnosability of the system and we reminded complexity results for verifying diagnosability of timed automata.

Chapter 4

Qualitative Abstractions for Hybrid Automata

In this chapter we elaborate abstractions of hybrid automata using principles from the qualitative reasoning domain. Abstractions as timed automata are presented. Algorithms for computing the abstractions are shown. Section 4.1 presents state-space decomposition abstractions of a hybrid automaton where timeless and timed abstractions are defined. Section 4.2 presents algorithms for automating the computation of the defined abstractions. Finally, section 4.3 applies the elaborated abstraction method to diagnosability verification by using a counter-example guided abstraction refinement loop.

4.1 Qualitative Abstractions

The language of hybrid automata as previously defined (Section 3.1) is not directly manageable (for the purpose of model checking for example). This is mainly due to the large expressiveness of the language. This fact translates mathematically by the differential equations having no closed form solutions in general. The challenging part about hybrid systems is to encode the continuous evolution into usable representations while taking into account the guard and reset regions. Consequently, the use of abstractions allows obtaining a simplified model, but nonetheless possibly sufficient and correct for formal properties verification. The decision at the abstract level can to some extent be inferred back to the concrete hybrid system. We first remind some abstraction principles and provide some intuitive

and motivating examples. We then define different abstraction schemes using and extending ideas from the qualitative reasoning domain. This is done by first introducing the finest abstractions of hybrid automata followed by decomposition of the hybrid state space into regions. We then define hybrid automata abstractions using a defined decomposition. We elaborate abstractions as discrete event systems, modeling reachability between the regions of the decomposition of the state space, and as timed automata, adding time constraints to the reachability relations. For the rest of the chapter, we consider a hybrid automaton H (Def.3.1, page 53).

4.1.1 Abstracting continuous change

Generally speaking, verifying basic properties such as reachability is undecidable for continuous systems. This is due in particular to the large expressiveness of continuous dynamics over the infinite domain of the reals [63]. A fortiori, for a hybrid system, a computation of the reachable set of states starting from an initial state is undecidable except for few classes [66]. An efficient practice is to partition infinite domains into a finite number of subsets, abstracting the system behavior in each of those subsets. In this section, we thus focus on abstractions that discretize the infinite state space defined by continuous variables into a finite set. The challenging part about abstractions is the choice made to select the representative sets and the criterion for choosing them. This choice relies entirely on the class of the properties one wishes to verify and on the structure and class of the hybrid system itself. Abstractions that can be refined are a necessary concern, as refinement allows if needed adding more information into the abstract system from the original one.

Sound and complete abstraction Figure 4.1 illustrates the exact reachable set (in grey) obtained from an initial square set X_0. Two abstractions are proposed (finite sets of squares within the bold lines). On the left is a **complete abstraction**, the exact reached set of states being contained within the abstraction squares. On the right is a **sound abstraction**, every square of the abstraction having a nonempty intersection with the exact reached set (in the example, it is

also complete, while the abstraction on the left is not sound). Take care that some authors use the opposite designation to qualify an abstraction.

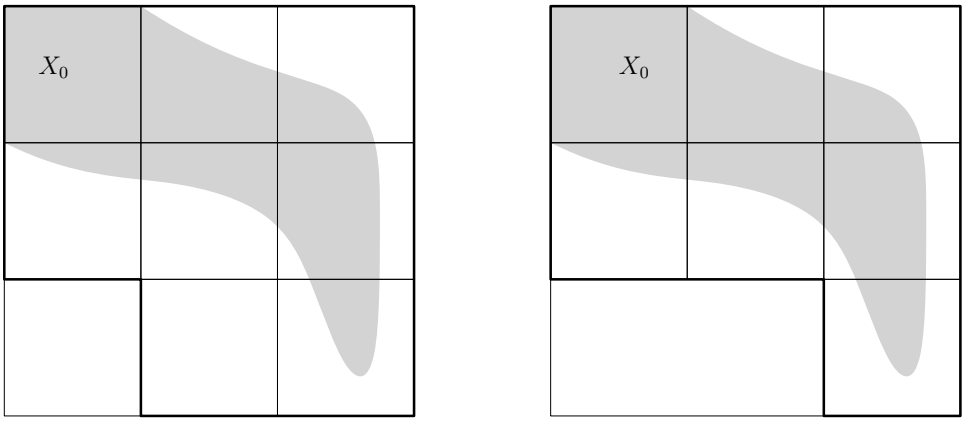


Figure 4.1: Square abstraction of a continuous evolution: complete unsound abstraction (left), sound and complete abstraction (right)

For a hybrid automaton H, the general idea is to use complete abstractions, i.e., whose behaviors encompass all concrete behaviors of H, for those properties we want to verify that are universally quantified on all the behaviors (the executions) of H, and to use sound abstractions, i.e., whose behaviors all abstract some concrete behavior of H, for those properties we want to verify that are existentially quantified on the behaviors of H. Both cases correspond in practice to over-approximations and under-approximations respectively. In fact to construct the complete unsound abstraction (left), it is sufficient to have computed an over-approximation of the reachable set of states, then any square representation that contains the over-approximation is a complete abstraction. While to construct the sound and complete abstraction (right), it is necessary to have an under-approximation of the reachable set of states, any square of the abstraction representation must guarantee that it has a non-empty intersection with the under-approximation set. Consequently, we will have the insurance that if the property is proved to be satisfied by the abstraction, then it is by H. If it is proved to be unsatisfiable at the abstraction level, then we will not be able to conclude in general and will resort on a refinement of the abstraction. In this thesis where we are particularly interested in the diagnosability property, we will use exclusively

complete abstractions (and thus almost always unsound, with so called spurious behaviors that do not reflect any concrete behavior) .

4.1.2 Abstractions using qualitative and invariant based reasoning

Using qualitative principles Let us apply qualitative based reasoning to abstract continuous change (reviewed in 2.2, page 17). Given a set of ODEs, the infinite state space of the continuous variables is discretized into a finite set. The discretization of $\mathbb{R}^n$ is often achieved by rectangles, i.e., is built by product from a discretization of $\mathbb{R}$. And this one is obtained by fixing a finite number of (rational) landmarks l_i, resulting in a finite partition in terms of open intervals (l_i, l_{i+1}) (with possibly infinite endpoints) and singleton intervals $[l_i]$, allowing what is called absolute order of magnitude reasoning. The coarsest partition (except $\mathbb{R}$ itself) is obtained from the single landmark 0 and corresponds to the sign partition: $(-\infty, 0)$, $[0]$, $(0, +\infty)$, giving rise to a partition of size 3^n of $\mathbb{R}^n$. It is particularly interesting when applied to the valuations of the variables derivatives, as it corresponds to discretize according to the sign of the derivative, which is constant within each set of the partition, and thus to the change direction of the variable itself (decreasing, constant, increasing). The variables being continuously differentiable, it is not possible for the sign of the derivative to pass from negative to positive without crossing zero. Exploiting this feature, we obtain a scheme of the behavior of the variables called "qualitative simulation" and obtain an overview of the system behavior [74, 53, 78].

Example 4.1. Consider this simple linear continuous system:

$$\begin{aligned} \dot{x} &= 3x \\ \dot{y} &= y - 1 \end{aligned} \tag{4.1}$$

Adopting the partition of the state space given by the signs of the derivatives, the abstract state space of size 9 is thus: $(\dot{x} > 0 \ \vee \ \dot{x} < 0 \ \vee \ \dot{x} = 0) \wedge (\dot{y} > 0 \ \vee \ \dot{y} < 0 \ \vee \ \dot{y} = 0)$. The transitions between the abstract states are computed according to the laws of evolution given the signs of the derivatives. The abstract state $(\dot{x} > 0 \ \wedge \ \dot{y} > 0)$ corresponds to the region $\{x, y \mid x > 0 \ \wedge \ y > 1\}$ in the

state space. From this state, no transition is possible to another abstract state. Suppose we wish to verify a basic reachability property: starting from the state $(1, 3)$ is it possible to reach the state $(-5, -4)$? The answer would be no, the proof is given using the previous abstraction method and inferring the property back to the original system. Such abstraction is complete: from any initial state (x_0, y_0) the solutions of the differential equation system 4.1 will always satisfy the constraints imposed by the abstract system rules, i.e., the possible transitions.

Abstractions for the verification of temporal properties The above abstraction is useful to trace the future evolution of the state given the initial one to prove a safety property of avoiding an unwanted state. However it is time-oblivious, time is not captured by the abstraction. Consequently, for proving a more complex property that involves the notion of time the above abstraction is not sufficient. One needs to add time as a separate state variable and correlate the variables changes to changes in time. A timed specification can be expressed using temporal logic.

Example 4.2. Let us reconsider the same dynamics of the previous example, suppose the initial set of states X_0 such that $X_0 = \{(x, y) \mid 1 < x < 2 \ \wedge \ 1 < y < 2 \ \wedge \ x < y\}$ and the property $F(x > y)$ where F is the "eventually" linear temporal logic (LTL) operator. Fp, where p is a Boolean proposition, is equivalent to $\exists t_0 \in \mathbb{R}^+, \forall t > t_0, p = true$. It is obvious that the rate at which x is increasing with respect to time is much larger than that of y. Hence, for all the initial states within X_0 the property is true. The previous abstraction method however does not capture the rate at which the derivative of x is changing and is thus useless for establishing the proof. Actually, changing the first equation in 4.1 by $\dot{x} = 0.5x$ would keep the abstract system unchanged and nevertheless change the truth value of the property. In our case, the system can be written as $\dot{\mathbf{x}} = A\mathbf{x} + \mathbf{b}$ where $\mathbf{x} = (x, y)^T$ and A is the corresponding matrix. We then deduce by computing the eigenvalues of , which are 3 and 1 in our example, that the rate at which x increases is larger than the rate at which y increases, which provides a sufficient proof that the above property holds when the system is initiated from X_0.

The previous example illustrates the case of a linear dynamics where the eigenvectors are not rotated by the linear transformation and are thus invariant for

the continuous system. Therefore, taking these two vectors into account during the abstraction process is an obvious choice. However, in the more general case of nonlinear dynamics, the invariant takes a more complex form. Some technique encodes the hybrid system, the property to verify and a specific parametric form of the invariant into an SMT (Satisfiability Modulo Theories) based solver and evaluates the unknown parameters of the invariant automatically. Once computed, the invariant is incorporated to make a finer and more representative abstraction [61].

4.1.3 The finest abstraction of a hybrid automaton

Let us consider the semantics of a hybrid automaton as an infinite transition system. It is the most granular abstraction achievable whose computation is not possible in practice. We will use the finest abstraction in the next section to define a less granular abstraction, that can be computed.

The timed transition system Let $\overline{S} = \bigcup_{q \in Q}(\{q\} \times Inv(q)) \subseteq S$ the (infinite) set of invariant satisfying states of H, $\overline{S}_0 = \bigcup_{q \in Q_0}(\{q\} \times Inv(q)) \subseteq S_0$ the subset of invariant satisfying initial states and $\rightarrow \subseteq \overline{S} \times L \times \overline{S}$ the transition relation defined by one or the other condition in Definition 3.2, page 57. The semantics of H is actually given by the labeled transition system $S^t_H = (\overline{S}, \overline{S}_0, L, \rightarrow)$, i.e., $[[H]]$ is the set of all paths of S^t_H issued from an initial state. S^t_H, called the **timed transition system of** H, is thus a discretization of H with infinite sets of states and of transition labels. It just abstracts continuous flows by timed transitions retaining only information about the source, the target and the duration of each flow and constitutes the finest abstraction of H we will consider.

The timeless transition system The timeless abstraction of S^t_H, called the **timeless transition system of** H, is obtained by ignoring also the duration of flows and thus defined as $S_H = (\overline{S}, \overline{S}_0, \Sigma \cup \{\epsilon\}, \rightarrow)$, obtained from S^t_H by replacing any timed transition $(q_i, \mathbf{x}_i) \xrightarrow{d_i} (q_{i+1}, \mathbf{x}_{i+1})$ with $d_i \in \mathbb{R}_+$ by the ϵ transition $(q_i, \mathbf{x}_i) \xrightarrow{\epsilon} (q_{i+1}, \mathbf{x}_{i+1})$, that can be considered as a silent transition. It has infinite set of states but finite set of transitions labels. It constitutes the finest timeless abstraction of H we will consider.

Proposition 4.1 (Correctness and completeness of the semantics)**.** *Any concrete behavior of H is timed (resp. timeless) abstracted into an $\overline{S}_0$ rooted path in S^t_H (resp. S_H). Conversely, any path in S^t_H (resp. S_H) that alternates continuous and discrete transitions (in particular any single transition) abstracts a part of a concrete behavior of H and, if the flow dynamics F is a singleton function (i.e., deterministic derivative), any $\overline{S}_0$ rooted path in S^t_H (resp. S_H) abstracts a concrete behavior of H. In this latter case, there is thus no spurious abstract behavior in S^t_H (resp. S_H), which expresses faithfully the behavior of H.*

4.1.4 Geometric decomposition of the state space

The finest abstraction cannot be computed in practice generally. Consequently, we present finite state space decomposition of a hybrid automaton. We will then present an abstraction based on different decompositions that incorporates reachability and time constraints. Later on, we will discuss the refinement of the abstraction yielding constraints with better precision than before refinement.

Definition 4.1 (Continuous space partition)**.** *A (finite) partition P of the Euclidean space $\mathbb{R}^n$ is a finite set of nonempty connected subsets of $\mathbb{R}^n$ such that every point $x \in \mathbb{R}^n$ is in one and only one of those subsets. We can write $\mathbb{R}^n = \biguplus_{p\in P} p$. An element p of P will be referred to as a partition element and we will call it a region. For a subset E of $\mathbb{R}^n$, we will denote by $P(E)$ the subset of regions of P that have a nonempty intersection with E.*

The only smoothness hypothesis we will impose for the moment over a partition is that any (finite) continuous path crosses only a finite number of times each region, more precisely, $\forall x : [0,1] \to \mathbb{R}^n$ a continuous function, $\forall p \in P$ a region, $x^{-1}(p)$ is a finite union of intervals. In practice, partitions are chosen enough regular and smooth, with regions in any dimension from n to 0 such as (from simpler to more complex) rectangles, zonotopes, polytopes or defined by a set of polynomial inequalities. The choice among the different partitions is guided by the property we wish to verify. For example, consider a continuous system with dynamics F. A coarse but helpful way to obtain a high level reachability mapping would be to identify regions of the state space that conserve the sign of F. E.g.,

in one dimension, for all elements of the same region the derivative signs would be all either negative or positive or null. Thus, the regions would be the connected components of the three subsets p_1, p_2, p_3 defined by:

$$p_1 = \{\mathbf{x} \in \mathbf{X} | \dot{x} > 0\}, p_2 = \{\mathbf{x} \in \mathbf{X} | \dot{x} < 0\}, p_3 = \{\mathbf{x} \in \mathbf{X} | \dot{x} = 0\} \quad (4.2)$$

For n dimensional systems, the regions would be the connected components of the 3^n subsets E_s of $\mathbb{R}^n$ parametrized by sign vectors $s \in \{-1, 0, +1\}^n$: $E_s = \{\mathbf{x} \in \mathbf{X} \mid \forall i, 1 \leq i \leq n,\ \dot{\mathbf{x}}^i < 0 \text{ if } s^i = -1,\ \dot{\mathbf{x}}^i = 0 \text{ if } s^i = 0,\ \dot{\mathbf{x}}^i > 0 \text{ if } s^i = +1\}$.

If the considered system is a hybrid automaton, it is practical to allow different partitions in different modes. In the following, we will assume that the sets $Init(q)$, $Inv(q)$, $G(\tau)$ and $R(\tau)(p)$ (for p connected subset of $G(\tau)$) can be expressed as finite unions of connected subsets (if this is not the case, we will over-approximate parts of them). We define thus a decomposition of the hybrid state space as follows.

Definition 4.2 (Hybrid state space decomposition)**.** *Given a hybrid automaton $H = (Q, X, S_0, \Sigma, F, \delta, Inv, G, R)$ and a set of partitions $\mathfrak{P}$, we say that $\mathfrak{P}$ decomposes H if there is a surjective function d such that $d : Q \to \mathfrak{P}$.*

The initial and invariant sets and the guards satisfiability domains and variables reset domains are primary elements to take into consideration while abstracting. For $q \in Q$ and $\tau = (q, \sigma, q') \in \delta$, we denote the regions families $d(q)(Init(q))$, $d(q)(Inv(q))$, $d(q)(G(\tau))$ by $d_{Init}(q)$, $d_{Inv}(q)$, $d_G(q, \tau) \subseteq d(q)$ and, for a region $p \in d_G(q, \tau)$, we denote $d(q')(R(\tau)(p \cap G(\tau)))$ by $d_R(q', \tau, p) \subseteq d(q')$. When possible, we will try to define d such that $Init(q)$, $Inv(q)$, $G(\tau)$ and $R(\tau)(p)$ are exactly the unions of the regions in those families (if not, those regions families over-approximate them).

4.1.5 Encoding hybrid automata reachability constraints

We have defined in 4.1.3 page 79, the timeless transition system S_H of a hybrid automaton H as the finest timeless abstraction that can be obtained. However, in practice S_H can only be computed for very restricted classes of hybrid automata. We will define a less granular time-abstract transition system based on a set of partitions and define the relations between adjacent regions.

Definition 4.3 (Adjacent regions)**.** *Two distinct regions p_1, p_2 of a partition P of $\mathbb{R}^n$ are adjacent if one intersects the boundary of the other: $p_1 \cap \overline{p_2} \neq \emptyset$ or $\overline{p_1} \cap p_2 \neq \emptyset$, where $\overline{p}$ refers to the closure of p.*

Definition 4.4 (Decomposition-based timeless abstract automaton of a hybrid automaton)**.** *Given a hybrid automaton $H = (Q, X, S_0, \Sigma, F, Inv, T)$ and a decomposition $(\mathfrak{P}, d)$ of H, we define the timeless abstract (finite) automaton of H with respect to $\mathfrak{P}$ as $DH_{\mathfrak{P}} = (Q_{DH}, Q_{0_{DH}}, \Sigma_{DH}, \delta_{DH})$ with:*

- $Q_{DH} = \{(q, p) | q \in Q, p \in d(q)\}$.
- $Q_{0_{DH}} = \{(q, p_{Init}) | q \in Q_0, p_{Init} \in d_{Init}(q)\}$.
- $\Sigma_{DH} = \Sigma \cup \{\epsilon\}$.
- $((q_i, p_k), \sigma, (q_j, p_l)) \in \delta_{DH}$ *iff one of both is true:*
  - $\sigma \in \Sigma$ *and* $p_k \in d_G(q_i, \tau)$ *and* $p_l \in d_R(q_j, \tau, p_k)$ *where* $\tau = (q_i, \sigma, q_j) \in \delta$.
  - $q_i = q_j$ *and* $\sigma = \epsilon$ *and* $p_k, p_l \in d_{Inv}(q_i)$ *are adjacent regions and* $\exists d \in \mathbb{R}_+^*$ *and* $\exists x : [0, d] \to \boldsymbol{X}$ *continuously differentiable function such that* $\forall t \in (0, d)\ \dot{x}(t) \in F(q_i, x(t))$, $\forall t \in [0, d]\ x(t) \in Inv(q_i)$, $x(0) \in p_k$, $x(d) \in p_l$, $\exists c\ 0 \leq c \leq d\ \forall t \in (0, c)\ x(t) \in p_k\ \forall t \in (c, d)\ x(t) \in p_l$ *and* $x(c) \in p_k \cup p_l$.

The defined timeless abstract automaton encodes reachability with adjacent regions of the state space, the events in Σ witnessing mode changes and ϵ transitions representing a continuous evolution between adjacent regions in the same mode. Notice that $((q_i, p_k), \sigma, (q_j, p_l)) \in \delta_{DH} \Rightarrow \exists \mathbf{x}_k \in p_k\ \exists \mathbf{x}_l \in p_l\ (q_i, \mathbf{x}_k) \xrightarrow{\sigma} (q_j, \mathbf{x}_l)$ in S_H, the converse being true for $\sigma \in \Sigma$. The mapping $\alpha_{\mathfrak{P}}$ defined by $\alpha_{\mathfrak{P}}((q, \mathbf{x})) = (q, p)$ with $p \in d(q)$ and $\mathbf{x} \in p$ defines an onto timeless abstraction function $\alpha_{\mathfrak{P}} : \overline{S} \to Q_{DH}$. If the flow condition F is a singleton, $\alpha_{\mathfrak{P}}$ maps any transition of S_H to a unique path in $DH_{\mathfrak{P}}$. The coarsest timeless abstract automaton is obtained when partitions of $\mathfrak{P}$ have all a unique region $p = \mathbf{X}$ and is thus (Q, Q_0, Σ, δ), i.e., the discrete part of H without its continuous part. It corresponds to the coarsest timeless abstraction function $\alpha_{\{\{\mathbf{X}\}\}}((q, \mathbf{x})) = q$. For our previous thermostat example, this gives $(\{off, on\}, \{off\}, \{B_{on}, B_{off}\}, \{(off, B_{on}, on),$

$(on, B_{off}, off)\}$) and the abstraction of the execution h given previously (3.1.4) is just $off \xrightarrow{B_{on}} on \xrightarrow{B_{off}} off \ldots$

Theorem 4.1 (Timeless abstraction completeness)**.** *Given a decomposition $\mathfrak{P}$ of H, any concrete behavior of H is timeless abstracted into a $Q_{0_{DH}}$ rooted path in $DH_{\mathfrak{P}}$ and any transition of $DH_{\mathfrak{P}}$ abstracts a part of a concrete behavior of H. If the flow condition F is a singleton function then the timeless abstraction function $\alpha_{\mathfrak{P}}$ defines a trace preserving mapping (still denoted by $\alpha_{\mathfrak{P}}$) from $\overline{S}_0$ rooted paths in S_H (i.e., timeless executions of H) to $Q_{0_{DH}}$ rooted paths in $DH_{\mathfrak{P}}$ and thus the language defined by S_H is included in the language defined by $DH_{\mathfrak{P}}$.*

Obviously, a path in $DH_{\mathfrak{P}}$ does not abstract in general a concrete behavior of H (as the behaviors parts abstracted by the individual transitions do not connect in general) which expresses that abstraction creates spurious behavior. Figure 4.2 illustrates a parallelogram abstraction. Two concrete trajectories h and h' are abstracted by two abstract paths respectively $p_1 \to p_2$ and $p_2 \to p_3$; however the abstract path $p_1 \to p_2 \to p_3$ admits no feasible concrete trajectory.

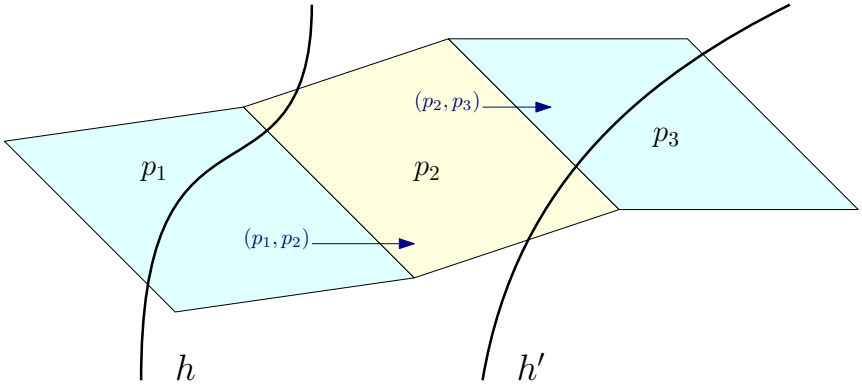


Figure 4.2: Abstraction spurious effect: every concrete trajectory is covered by an abstract trajectory but the opposite does not necessarily hold.

Abstracting with partial observations If now H is a POHA, in the same way we defined the observation of a concrete execution in Definition 3.13 we define the observation of its timeless abstraction.

Definition 4.5 (Timeless abstraction observation)**.** *Given a POHA H and $h = (q_0, p_0) \xrightarrow{\sigma_0} (q_1, p_1)...(q_i, p_i) \xrightarrow{\sigma_i} ...$, with $\sigma_i \in \Sigma \cup \{\epsilon\}$, a timeless abstract path in $DH_{\mathfrak{P}}$, the observation of h is defined as $Obs(h) = p_0^o, \sigma_0^o, p_1^o...p_i^o, \sigma_i^o, ...$, where*

- *p_i^o is obtained by projecting p_i on variables in X_o.*
- *$\sigma_i^o = \sigma_i$ if $\sigma_i \in \Sigma_o$. Otherwise, $\sigma_i^o = \varepsilon$, which is then removed from $Obs(h)$.*

Consider an execution $h \in [[H]]$ of the POHA H, $h = (q_0, \mathbf{x}_0) \xrightarrow{l_0} (q_1, \mathbf{x}_1)...(q_i, \mathbf{x}_i) \xrightarrow{l_i} ...$, with $l_i \in \Sigma \cup \mathbb{R}_+$, its (timed) observation $Obs(h) = \mathbf{x}_0^o, l_0^o, \mathbf{x}_1^o...\mathbf{x}_i^o, l_i^o, ...$ as in Definition 3.13, its timeless abstraction $\alpha_{\mathfrak{P}}(h) = (q_0, p_0) \xrightarrow{\sigma_0} (q_1, p_1)...(q_i, p_i) \xrightarrow{\sigma_i} ...$, with $\sigma_i \in \Sigma \cup \{\epsilon\}$, as in Theorem 4.1 (assuming F a singleton) and the observation of this last one $Obs(\alpha_{\mathfrak{P}}(h)) = p_0^o, \sigma_0^o, p_1^o...p_i^o, \sigma_i^o, ...$ as in Definition 4.5. We could try to define the timeless abstraction of the observation $Obs(h)$. A natural definition would be $\alpha_{\mathfrak{P}}(Obs(h)) = p_0', \sigma_0^o, p_1'...p_i', \sigma_i^o, ...$, with $\sigma_i^o = l_i^o$ if $l_i^o \in \Sigma_o$ (and $= \epsilon$, which is removed, otherwise), i.e., the same σ_i^o's as in $Obs(\alpha_{\mathfrak{P}}(h))$, and $p_i' = \biguplus_{\{p | \mathbf{x}_i^o \in p^o\}} p^o$ the union of the projections on X_o of all regions containing a value whose projection on X_o is equal to $\mathbf{x}_i^o$ (assuming to simplify the same partition for each mode, as the mode may be unknown from observation). So, we notice that $Obs(\alpha_{\mathfrak{P}}(h))$ is more precise than $\alpha_{\mathfrak{P}}(Obs(h))$, as $p_i^o \subseteq p_i'$, which we denote by $Obs(\alpha_{\mathfrak{P}}(h)) \sqsubseteq \alpha_{\mathfrak{P}}(Obs(h))$ to mean that both sequences have common events and there is inclusion of the qualitative space values as subsets of $\mathbf{X}_o$, the valuations set corresponding to the observable variables.

4.1.6 Encoding hybrid automata time constraints

We are concerned with verifying temporal properties of hybrid systems and checking the diagnosability property using time constraints. For this reason, we define in this subsection, always related to a decomposition of the state space into partitions, an abstraction of the hybrid automaton as a timed automaton that partly captures the time constraints at the level of the regions. We will first introduce some intuitive ideas. Consider a partition P of the $\mathbb{R}^n$ state space of a continuous system with arbitrary dynamics F, the set of trajectories (i.e., the continuous solution flows) entering a region $p \in P$ is in one of these two cases: either at least one of the trajectories ends up trapped inside p for all future times or all of them exit

p to an adjacent region within a bounded time under the continuity assumption. In the first case, no time constraint can be associated with the region p unless a reshaping of p is applied; in the latter, it is possible to compute time constraints satisfied by all trajectories entering and leaving the region p. We will give a formal definition of the timed automaton constructed from given hybrid automaton and partitions set and then discuss some cases where a time bound can be practically computed.

Definition 4.6 (Region time interval and time bounds)**.** *Given a continuous system CS, a partition P of $\mathbb{R}^n$ and $p \in P$ one of its regions, we say that $I_p = [t_{min}, t_{max}]$, with $t_{min}, t_{max} \in \mathbb{R}_+ \cup \{+\infty\}$, is a region time interval of p for CS if all trajectories of the CS entering p at time t leave p at time $t + t_{min}$ at least and $t + t_{max}$ at most. t_{min} and t_{max} are lower and upper time bounds of p. For a hybrid automaton, we denote the time interval relative to the region p in mode q as $I_{(q,p)}$.*

Definition 4.7 (Decomposition-based timed abstract automaton of a hybrid automaton)**.** *Given a hybrid automaton $H = (Q, X, S_0, \Sigma, F, \delta, Inv, G, R)$, a decomposition $(\mathfrak{P}, d)$ and the timeless abstract automaton $DH_{\mathfrak{P}} = (Q_{DH}, Q_{0_{DH}}, \Sigma_{DH}, \delta_{DH})$ of H with respect to $\mathfrak{P}$, we define the timed abstract automaton of H with respect to $\mathfrak{P}$ as $TH_{\mathfrak{P}} = (Q_{DH}, \{c\}, Q_{0_{DH}}, \Sigma_{DH}, Inv_{TH}, (\delta_{DH}, G_{TH}, Y_{TH}))$ such that, $\forall (q, p) \in Q_{DH}$ with a region time interval $I_{(q,p)} = [t_{min}, t_{max}]$:*

- $Inv_{TH}((q, p)) = [0, t_{max}]$.
- $\forall \tau = ((q, p), \sigma, (q_1, p_1)) \in \delta_{DH}$, $G_{TH}(\tau) = [t_{min}, +\infty)$ *if* $\sigma = \epsilon$ *and* p *does not intersect any reset set (i.e.,* $\forall \tau' = (q', \sigma', q) \in \delta_{DH}\ p \notin d(q)(R(\tau')(G(\tau'))))$ *or* $[0, +\infty)$ *else.*

and, $\forall \tau \in \delta_{TH}$, $Y_{TH}(\tau) = \{c\}$.

The timed abstract automaton adds time constraints to those states (q, p) of the timeless abstract automaton for which an interval $I_{(q,p)}$ is computable as non-trivial (i.e., $I_{(q,p)} \neq [0, +\infty)$), by using one local clock c (reset at 0 in each state) that measures the sojourn duration t in each state (q, p), i.e., in each region p, and coding these constraints by means of invariant and guard of c in each state.

The invariant codes the maximum sojourn duration as the upper time bound of the region p and the guard codes the minimum sojourn duration as the lower time bound of the region p when both entering and leaving the region are not the result of discrete jumps (controlled here directly for the out-transition and by requiring that p does not intersect any reset set for all possible in-transitions). In the thermostat example, consider the partition into two regions associated to the mode *off* given by the initial states set $(\mathit{off}, [80, 90])$ and by $(\mathit{off}, [68, 80))$. Then we take as time bounds for $(\mathit{off}, [80, 90])$ $t_{min} = 0$ and $t_{max} = 0.12$ (the exact upper bound, i.e., the time for the temperature to decrease from 90 to 80 is $Log(\frac{9}{8})$). It means that we define in the timed abstract automaton $Inv_{TH}((\mathit{off}, [80, 90])) = [0, 0.12]$. A beginning of execution of the timed abstract automaton is, for example $(\mathit{off}, [80, 90]) \xrightarrow{0.08} (\mathit{off}, [68, 80])$.

Theorem 4.2 (Timed abstraction completeness)**.** *Given a decomposition $\mathfrak{P}$ of H, any concrete behavior of H is timed abstracted into an execution in $TH_{\mathfrak{P}}$. If the flow condition F is a singleton function then the abstraction function $\alpha_{\mathfrak{P}}$ defines a mapping, denoted by $\alpha^t_{\mathfrak{P}}$, from $\overline{S}_0$ rooted paths in S^t_H (i.e., executions of H) to executions in $TH_{\mathfrak{P}}$. This mapping is trace preserving once ϵ labels are erased from executions traces in $TH_{\mathfrak{P}}$ and time period labels are added up between two consecutive events labels in both executions traces in S^t_H and in $TH_{\mathfrak{P}}$. This means that, for any execution $(q_0, \boldsymbol{x}_0) \xrightarrow{w}_* (q_i, \boldsymbol{x}_i) \in [[H]]$, with $w \in L^*$ (where $L = \Sigma \cup \mathbb{R}_+$), it exists a unique execution $(q_0, p_0) \xrightarrow{w'}_* (q_j, p_j) \in [[TH_{\mathfrak{P}}]]$, with $w' \in L'^*$ (where $L' = L \cup \{\epsilon\}$), $\boldsymbol{x}_0 \in p_0$, $q_j = q_i$, $\boldsymbol{x}_i \in p_j$, $w'_{|\Sigma} = w_{|\Sigma}$ (where $_{|\Sigma}$ is the projection of timed words on words on Σ^*) and, for any two successive events $w_l = w'_{l'}$ and $w_m = w'_{m'}$ of $w_{|\Sigma}$, $\sum_{l'<k'<m', w'_{k'} \neq \epsilon} w'_{k'} = \sum_{l<k<m} w_k$.*

Forgetting time, i.e., removing the clock, provides a natural abstraction function α from $TH_{\mathfrak{P}}$ to $DH_{\mathfrak{P}}$ which maps an execution $(q_0, p_0) \xrightarrow{l_0} (q_1, p_1)...(q_i, p_i) \xrightarrow{l_i}$ ..., with $l_i \in \Sigma \cup \{\epsilon\} \cup \mathbb{R}_+$, in $TH_{\mathfrak{P}}$ into the execution $(q_0, p_0) \xrightarrow{\sigma_0} (q_1, p_1)...(q_i, p_i) \xrightarrow{\sigma_i}$ ..., with $\sigma_i \in \Sigma \cup \{\epsilon\}$, in $DH_{\mathfrak{P}}$, with $\sigma_i = l_i$ if $l_i \in \Sigma \cup \{\epsilon\}$ and continuous transitions labeled by $l_i = d_i \in \mathbb{R}_+$ are suppressed. We have: $\alpha_{\mathfrak{P}} = \alpha \circ \alpha^t_{\mathfrak{P}}$.

Definition 4.8 (Timed abstraction observation)**.** *Given a POHA H and $h = (q_0, p_0) \xrightarrow{l_0} (q_1, p_1)...(q_i, p_i) \xrightarrow{l_i}$ ..., with $l_i \in \Sigma \cup \{\epsilon\} \cup \mathbb{R}_+$, an execution in $TH_{\mathfrak{P}}$, i.e., a*

timed abstract path, the observation of h is defined as $Obs(h) = p_0^o, l_0^o, p_1^o ... p_i^o, l_i^o, ...$, where

- *p_i^o is obtained by projecting p_i on variables in X_o.*
- *$l_i^o = l_i$ if $l_i \in \Sigma_o \cup \mathbb{R}_+$. Otherwise, $l_i^o = \varepsilon$, which is then removed from $Obs(h)$.*

As for the timeless case, we can define the timed abstraction of an observation (of an execution $h \in [[H]]$) and we obtain: $Obs(\alpha^t_{\mathfrak{P}}(h)) \sqsubseteq \alpha^t_{\mathfrak{P}}(Obs(h))$.
From another side, the abstraction function α that forgets time maps a timed abstract observation $p_0^o, l_0^o, p_1^o ... p_i^o, l_i^o, ...$, with $l_i^o \in \Sigma_o \cup \mathbb{R}_+$, into the timeless abstract observation $p_0^o, \sigma_0^o, p_1^o ... p_i^o, \sigma_i^o, ...$, with $\sigma_i^o \in \Sigma_o$, suppressing duration labels $l_i = d_i \in \mathbb{R}_+$. For any concrete execution $h \in [[H]]$, we have: $Obs(\alpha_{\mathfrak{P}}(h)) = \alpha(Obs(\alpha^t_{\mathfrak{P}}(h)))$, i.e., $Obs \circ \alpha_{\mathfrak{P}} = \alpha \circ Obs \circ \alpha^t_{\mathfrak{P}}$.

We now formally define a refinement operation of the previously defined abstraction. For this purpose, we construct a finer couple of discrete and timed automata by defining a more granular decomposition for regions and give the necessary assumptions to compute such refinement. By making the partition more granular in regions of interest, tighter time bounds are also obtained. The refinement is a necessary step when a proof for the verification of a property could not be made at a given abstraction level.

4.1.7 Refining the partitioning of the state space

Definition 4.9 (Partition refinement)**.** *Given two partitions P and P' of $\mathbb{R}^n$, we say that P' is a refining partition of P iff $\forall p' \in P'\ \exists p \in P\ p' \subseteq p$. This implies: $\forall p \in P\ \exists P'_p \subseteq P'\ p = \biguplus_{p' \in P'_p} p'$.*

Definition 4.10 (Hybrid state space decomposition refinement)**.** *Given two decompositions $(\mathfrak{P}, d)$ and $(\mathfrak{P}', d')$ of a hybrid automaton $H = (Q, X, S_0, \Sigma, F, Inv, T)$, we say that $\mathfrak{P}'$ refines $\mathfrak{P}$, denoted by $\mathfrak{P}' \preceq \mathfrak{P}$, if $\forall q \in Q\ d'(q)$ is a refining partition of $d(q)$.*

4.1.8 Refined timeless model

Definition 4.11 (Refined timeless abstract automaton)**.** *Given a hybrid automaton H and two abstract timeless automata $DH_{\mathfrak{P}}$ and $DH_{\mathfrak{P}'}$ of H with respect to two decompositions $(\mathfrak{P}, d)$ and $(\mathfrak{P}', d')$ respectively, we say that $DH_{\mathfrak{P}'}$ is a timeless refinement of $DH_{\mathfrak{P}}$ abstracting H if $\mathfrak{P}' \prec \mathfrak{P}$, which we denote by $DH_{\mathfrak{P}'} \prec DH_{\mathfrak{P}}$.*

Definition 4.12 (State split operation)**.** *Given an abstract timeless automaton $DH_{\mathfrak{P}}$ of a hybrid automaton H, a split operation of the state $(q, p) \in Q_{DH}$ is defined by a partition $\{p_1, p_2\}$ of p, $p = p_1 \biguplus p_2$, and results in two states (q, p_1) and (q, p_2) and in the refined abstract timeless automaton $DH_{\mathfrak{P}'}$ with $\mathfrak{P}'$ obtained from $\mathfrak{P}$ by replacing $d(q)$ by $d'(q) = d(q) \backslash \{p\} \cup \{p_1, p_2\}$.*

The construction of $DH_{\mathfrak{P}'}$ from $DH_{\mathfrak{P}}$ after a (q, p) state split is a local operation as only the transitions of δ_{DH} having as source or as destination the state (q, p) have to be recomputed from H. In practice, the refined model is obtained by performing a finite number of state split operations. After having performed the split operations and in order for the obtained automaton to satisfy definition 4.4, it is only required to recompute some of its transitions, while inheriting the rest from $DH_{\mathfrak{P}}$. Let $Q_{split} \subseteq Q_{DH}$ be the set of split states and $post(Q_{split}) = \{q \in Q_{DH} \mid \exists q_s \in Q_{split} \exists (q_s, \sigma, q) \in \delta_{DH}\}$ and $pre(Q_{split}) = \{q \in Q_{DH} \mid \exists q_s \in Q_{split} \exists (q, \sigma, q_s) \in \delta_{DH}\}$. Then to obtain $DH_{\mathfrak{P}'}$ it is sufficient to only recompute transitions (q, σ, q') such that $q, q' \in Q_{split} \cup post(Q_{split}) \cup pre(Q_{split})$.

The onto abstraction function $\alpha_{\mathfrak{P}',\mathfrak{P}} : Q_{DH}^{\mathfrak{P}'} \to Q_{DH}^{\mathfrak{P}}$ defined by $\alpha_{\mathfrak{P}',\mathfrak{P}}((q, p')) = (q, p)$ with $p' \subseteq p$ defines a trace preserving mapping $\alpha_{\mathfrak{P}',\mathfrak{P}}$ from $DH_{\mathfrak{P}'}$ to $DH_{\mathfrak{P}}$ (and thus the language defined by $DH_{\mathfrak{P}'}$ is included in the language defined by $DH_{\mathfrak{P}}$) and we have: $\alpha_{\mathfrak{P}} = \alpha_{\mathfrak{P}',\mathfrak{P}} \circ \alpha_{\mathfrak{P}'}$. Defining in a natural way as previously the $\mathfrak{P}$-abstraction $\alpha_{\mathfrak{P}',\mathfrak{P}}$ of the observation of a timeless $\mathfrak{P}'$-abstract execution h, we obtain: $Obs(\alpha_{\mathfrak{P}',\mathfrak{P}}(h)) \sqsubseteq \alpha_{\mathfrak{P}',\mathfrak{P}}(Obs(h))$.

4.1.9 Refined timed model

Definition 4.13 (Refined timed abstract automaton)**.** *Given a hybrid automaton H and two abstract timed automata $TH_{\mathfrak{P}}$ and $TH_{\mathfrak{P}'}$ of H with respect to two*

decompositions $(\mathfrak{P}, d)$ *and* $(\mathfrak{P}', d')$ *respectively, we say that* $TH_{\mathfrak{P}'}$ *is a timed refinement of* $TH_{\mathfrak{P}}$ *abstracting* H *if* $\mathfrak{P}' \prec \mathfrak{P}$. *We denote it similarly by* $TH_{\mathfrak{P}'} \prec TH_{\mathfrak{P}}$.

Concerning the refined abstract timed automaton $TH_{\mathfrak{P}'}$ resulting from a split of (q, p) into (q, p_1) and (q, p_2), if $I_{(q,p)} = [t_{min}, t_{max}]$ the region time intervals $I_{(q,p_1)} = I_{(q,p_2)} = [0, t_{max}]$ can be adopted in first approximation as they are safe, but in general new tighter time bounds are recomputed from H for the sojourn duration in the regions p_1 and p_2. Thus, the refined timed model is obtained by a finite sequence of the two operations:

- **State split**: similar as before, the state split of (q, p) whose time interval is $I_{(q,p)} = [t_{min}, t_{max}]$ yields (q, p_1) and (q, p_2). The time intervals for the new split regions are set as $I_{(q,p_1)} = I_{(q,p_2)} = [0, t_{max}]$ (t_{max} stays a safe upper bound of the sojourn duration but t_{min} is reset to 0 since the split induces a distance shrink).

- **Time bounds refinement:** in this case, more precise time bounds are obtained for a given region of a discrete state, i.e., if $I_{(q,p)} = [t_{min}, t_{max}]$ then $I'_{(q,p)} = [t'_{min}, t'_{max}]$ with $t'_{min} \geq t_{min}$ and $t'_{max} \leq t_{max}$, at least one of both being a strict inequality.

The onto abstraction function $\alpha_{\mathfrak{P}',\mathfrak{P}}$ defines a trace preserving mapping $\alpha^t_{\mathfrak{P}',\mathfrak{P}}$ from $TH_{\mathfrak{P}'}$ to $TH_{\mathfrak{P}}$, after trace simplification as in Theorem 4.2 and provided the time bounds used in $TH_{\mathfrak{P}'}$, once added for all regions p' included in a given region p, are at least as tight as the time bounds used in $TH_{\mathfrak{P}}$. And we have: $\alpha^t_{\mathfrak{P}} = \alpha^t_{\mathfrak{P}',\mathfrak{P}} \circ \alpha^t_{\mathfrak{P}'}$. Finally, for any timed $\mathfrak{P}'$-abstract execution h, we obtain: $Obs(\alpha^t_{\mathfrak{P}',\mathfrak{P}}(h)) \sqsubseteq \alpha^t_{\mathfrak{P}',\mathfrak{P}}(Obs(h))$.

4.2 Algorithmic computation of the abstraction

In the previous section we formally defined timeless and timed abstractions of a given hybrid automaton based on space partitioning. In this section, we present algorithmic steps to compute the timeless abstraction. Then, we discuss methods for computing the bounds of the timed abstraction. The next chapter will present a tool that applies the presented algorithms together with experimental results.

The timeless abstraction computation steps are summarized as follows and detailed next:

- For each mode q of H, compute a symbolic representation of the abstract states using qualitative reasoning rules.
- For each computed symbolic state st, evaluate the presence of a transition towards abstract states whose concretization is adjacent to the concretization of st.
- Apply a linking operation to the mode abstractions taking into account guards and resets.

For the rest of the thesis, we will instantiate the hybrid automata class to polynomial hybrid automata (page 65, Definition 3.11), hence we hold the following assumption:

Assumption 4.1. We assume that all dynamics, initial sets, invariant sets, guards and resets of the considered hybrid automaton are given by polynomials in the $\mathbb{Q}[X]$ ring.

Example 4.3 (Two dimensional polynomial hybrid automaton)**.** To illustrate the computation steps of this section we will consider the following example throughout this section. The example is purely theoretical and does not reflect a practical application.

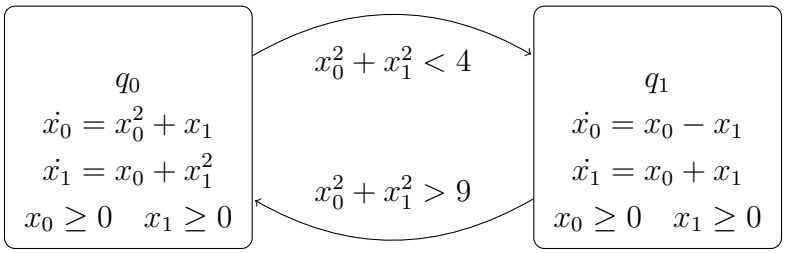


Figure 4.3: Polynomial hybrid automaton example

4.2.1 Computing qualitative partitions

Partitioning rules Consider a mode q of H. We will apply the geometric decomposition seen in subsection 4.1.4 to define $d(q)$ and compute the abstraction $DH_{\{d(q)\}}$ of q. Thus, the regions are the connected components of the 3^n subsets E_s of $\mathbb{R}^n$ parametrized by sign vectors $s \in \{-1, 0, +1\}^n$: $E_s = \{\mathbf{x} \in \mathbf{X} \mid \forall i, 1 \leq i \leq n,\ \dot{\mathbf{x}}^i < 0 \text{ if } s^i = -1,\ \dot{\mathbf{x}}^i = 0 \text{ if } s^i = 0,\ \dot{\mathbf{x}}^i > 0 \text{ if } s^i = +1\}$. Note that in the case of polynomial dynamics, the sets conserving the sign vector are **not necessarily connected**, in such case we will reason over non-connected sets and use invariants ($Inv(q)$) to separate them. To further partition algorithmically the state-space into connected sets we refer the reader to algorithms in [12]. Moreover such sets are **not necessarily convex**. Nonetheless, for dynamics given by linear differential equations, the sign conserving sets are connected and convex.

Expressing abstract states. We use semi-algebraic sets to represent regions using (unions of) (in)equalities of polynomials over X. For example, if X contains two variables, we can express a circular set as: $(x - x_0)^2 + (y - y_0)^2 < r^2$ with x_0, y_0 and r as given rational numbers.

Definition 4.14. *A semi-algebraic set in $\mathbb{R}^n$ is a finite union of sets defined each one by a finite number of equalities or inequalities over polynomials.*

Computing the abstract states The computation is performed in the following way. For a mode q of the hybrid automaton, a stack $Stack(q)$ is initialized with polynomials representing the initial and guard sets, incoming resets and dynamics. The stack is then extended as follows: each polynomial from the stack is copied three times and copies are assigned $+, -$ and *null* symbols. The regions expressions are obtained by symbolically computing the cross product of every element in the stack with all the other elements corresponding to different polynomials. The result is the set of sign-conserving semi-algebraic sets. The abstract states are then labeled whether they belong to the initial set, guard condition or reset, or none of the above. The first computation step is symbolic, thus, some regions are not feasible (i.e., are empty sets). A first check is performed to eliminate all empty regions. This can be implemented by finding whether or not a set of polynomial

equalities and inequalities admits a solution. The evaluation is performed using a quantifier elimination solver applying cylindrical algebraic decomposition and will be discussed in the next subsection. Abstract states whose region representation admits no solution are simply discarded from the stack.

Example 4.4 (Computing an initial qualitative partition)**.** Applying the previous steps for the considered example (Example 4.3, page 90), the initial partition of mode q_0 (whose null-clines and guard boundary are shown on Figure 4.4) contains the abstract states represented in Table 4.1. The empty regions check eliminates 8 regions and we are left with 18 abstract states (Table 4.2). When limiting the study zone to the specified invariant of q_0, $Inv(q_0) = \{x_0 \geq 0, x_1 \geq 0\}$, we are left with three states (Table 4.3).

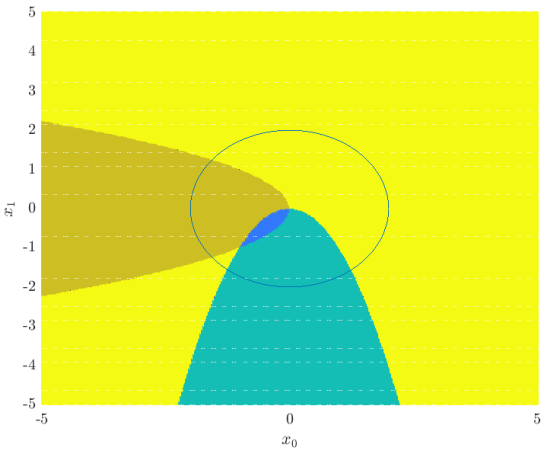


Figure 4.4: Qualitative partition of mode q_0

Abstract States					
S_0	$x_0^2 + x_1 = 0$ $\wedge\ x_0 + x_1^2 = 0$ $\wedge\ x_0^2 + x_1^2 = 4$	S_1	$x_0^2 + x_1 = 0$ $\wedge\ x_0 + x_1^2 = 0$ $\wedge\ x_0^2 + x_1^2 < 4$	S_2	$x_0^2 + x_1 = 0$ $\wedge\ x_0 + x_1^2 = 0$ $\wedge\ x_0^2 + x_1^2 > 4$
S_3	$x_0^2 + x_1 = 0$ $\wedge\ x_0 + x_1^2 < 0$ $\wedge\ x_0^2 + x_1^2 = 4$	S_4	$x_0^2 + x_1 = 0$ $\wedge\ x_0 + x_1^2 < 0$ $\wedge\ x_0^2 + x_1^2 < 4$	S_5	$x_0^2 + x_1 = 0$ $\wedge\ x_0 + x_1^2 < 0$ $\wedge\ x_0^2 + x_1^2 > 4$
S_6	$x_0^2 + x_1 = 0$ $\wedge\ x_0 + x_1^2 > 0$ $\wedge\ x_0^2 + x_1^2 = 4$	S_7	$x_0^2 + x_1 = 0$ $\wedge\ x_0 + x_1^2 > 0$ $\wedge\ x_0^2 + x_1^2 < 4$	S_8	$x_0^2 + x_1 = 0$ $\wedge\ x_0 + x_1^2 > 0$ $\wedge\ x_0^2 + x_1^2 > 4$
S_9	$x_0^2 + x_1 < 0$ $\wedge\ x_0 + x_1^2 = 0$ $\wedge\ x_0^2 + x_1^2 = 4$	S_{10}	$x_0^2 + x_1 < 0$ $\wedge\ x_0 + x_1^2 = 0$ $\wedge\ x_0^2 + x_1^2 < 4$	S_{11}	$x_0^2 + x_1 < 0$ $\wedge\ x_0 + x_1^2 = 0$ $\wedge\ x_0^2 + x_1^2 > 4$
S_{12}	$x_0^2 + x_1 < 0$ $\wedge\ x_0 + x_1^2 < 0$ $\wedge\ x_0^2 + x_1^2 = 4$	S_{13}	$x_0^2 + x_1 < 0$ $\wedge\ x_0 + x_1^2 < 0$ $\wedge\ x_0^2 + x_1^2 < 4$	S_{14}	$x_0^2 + x_1 < 0$ $\wedge\ x_0 + x_1^2 < 0$ $\wedge\ x_0^2 + x_1^2 > 4$
S_{15}	$x_0^2 + x_1 < 0$ $\wedge\ x_0 + x_1^2 > 0$ $\wedge\ x_0^2 + x_1^2 = 4$	S_{16}	$x_0^2 + x_1 < 0$ $\wedge\ x_0 + x_1^2 > 0$ $\wedge\ x_0^2 + x_1^2 < 4$	S_{17}	$x_0^2 + x_1 < 0$ $\wedge\ x_0 + x_1^2 > 0$ $\wedge\ x_0^2 + x_1^2 > 4$
S_{18}	$x_0^2 + x_1 > 0$ $\wedge\ x_0 + x_1^2 = 0$ $\wedge\ x_0^2 + x_1^2 = 4$	S_{19}	$x_0^2 + x_1 > 0$ $\wedge\ x_0 + x_1^2 = 0$ $\wedge\ x_0^2 + x_1^2 < 4$	S_{20}	$x_0^2 + x_1 > 0$ $\wedge\ x_0 + x_1^2 = 0$ $\wedge\ x_0^2 + x_1^2 > 4$
S_{21}	$x_0^2 + x_1 > 0$ $\wedge\ x_0 + x_1^2 < 0$ $\wedge\ x_0^2 + x_1^2 = 4$	S_{22}	$x_0^2 + x_1 > 0$ $\wedge\ x_0 + x_1^2 < 0$ $\wedge\ x_0^2 + x_1^2 < 4$	S_{23}	$x_0^2 + x_1 > 0$ $\wedge\ x_0 + x_1^2 < 0$ $\wedge\ x_0^2 + x_1^2 > 4$
S_{24}	$x_0^2 + x_1 > 0$ $\wedge\ x_0 + x_1^2 > 0$ $\wedge\ x_0^2 + x_1^2 = 4$	S_{25}	$x_0^2 + x_1 > 0$ $\wedge\ x_0 + x_1^2 > 0$ $\wedge\ x_0^2 + x_1^2 < 4$	S_{26}	$x_0^2 + x_1 > 0$ $\wedge\ x_0 + x_1^2 > 0$ $\wedge\ x_0^2 + x_1^2 > 4$

Table 4.1: Qualitative abstract states for mode q_0 of Example 4.3

Non-empty Abstract States					
S_0	$x_0^2 + x_1 < 0$ $\wedge\ x_0 + x_1^2 < 0$ $\wedge\ x_0^2 + x_1^2 < 4$	S_1	$x_0^2 + x_1 < 0$ $\wedge\ x_0 + x_1^2 > 0$ $\wedge\ x_0^2 + x_1^2 > 4$	S_2	$x_0^2 + x_1 < 0$ $\wedge\ x_0 + x_1^2 > 0$ $\wedge\ x_0^2 + x_1^2 < 4$
S_3	$x_0^2 + x_1 < 0$ $\wedge\ x_0 + x_1^2 > 0$ $\wedge\ x_0^2 + x_1^2 = 4$	S_4	$x_0^2 + x_1 < 0$ $\wedge\ x_0 + x_1^2 = 0$ $\wedge\ x_0^2 + x_1^2 < 4$	S_5	$x_0^2 + x_1 > 0$ $\wedge\ x_0 + x_1^2 < 0$ $\wedge\ x_0^2 + x_1^2 > 4$
S_6	$x_0^2 + x_1 > 0$ $\wedge\ x_0 + x_1^2 < 0$ $\wedge\ x_0^2 + x_1^2 < 4$	S_7	$x_0^2 + x_1 > 0$ $\wedge\ x_0 + x_1^2 < 0$ $\wedge\ x_0^2 + x_1^2 = 4$	S_8	$x_0^2 + x_1 > 0$ $\wedge\ x_0 + x_1^2 > 0$ $\wedge\ x_0^2 + x_1^2 > 4$
S_9	$x_0^2 + x_1 > 0$ $\wedge\ x_0 + x_1^2 > 0$ $\wedge\ x_0^2 + x_1^2 < 4$	S_{10}	$x_0^2 + x_1 > 0$ $\wedge\ x_0 + x_1^2 > 0$ $\wedge\ x_0^2 + x_1^2 = 4$	S_{11}	$x_0^2 + x_1 > 0$ $\wedge\ x_0 + x_1^2 = 0$ $\wedge\ x_0^2 + x_1^2 > 4$
S_{12}	$x_0^2 + x_1 > 0$ $\wedge\ x_0 + x_1^2 = 0$ $\wedge\ x_0^2 + x_1^2 < 4$	S_{13}	$x_0^2 + x_1 > 0$ $\wedge\ x_0 + x_1^2 = 0$ $\wedge\ x_0^2 + x_1^2 = 4$	S_{14}	$x_0^2 + x_1 = 0$ $\wedge\ x_0 + x_1^2 < 0$ $\wedge\ x_0^2 + x_1^2 < 4$
S_{15}	$x_0^2 + x_1 = 0$ $\wedge\ x_0 + x_1^2 > 0$ $\wedge\ x_0^2 + x_1^2 > 4$	S_{16}	$x_0^2 + x_1 = 0$ $\wedge\ x_0 + x_1^2 > 0$ $\wedge\ x_0^2 + x_1^2 < 4$	S_{17}	$x_0^2 + x_1 = 0$ $\wedge\ x_0 + x_1^2 > 0$ $\wedge\ x_0^2 + x_1^2 = 4$
S_{18}	$x_0^2 + x_1 = 0$ $\wedge\ x_0 + x_1^2 = 0$ $\wedge\ x_0^2 + x_1^2 < 4$				

Table 4.2: Qualitative abstract states after empty regions check

Non-empty Abstract States					
S_8	$x_0^2 + x_1 > 0$ $\wedge\ x_0 + x_1^2 > 0$ $\wedge\ x_0^2 + x_1^2 > 4$	S_9	$x_0^2 + x_1 > 0$ $\wedge\ x_0 + x_1^2 > 0$ $\wedge\ x_0^2 + x_1^2 < 4$	S_{10}	$x_0^2 + x_1 > 0$ $\wedge\ x_0 + x_1^2 > 0$ $\wedge\ x_0^2 + x_1^2 = 4$

Table 4.3: Non-empty invariant satisfying qualitative abstract states

4.2.2 Computing transitions between regions

We show in this section how to compute transitions between abstract states. The absence of incoming transitions to an abstract state translates by the absence of incoming trajectories to the region on the concrete level. The presence of a transition from abstract state S to S' translates by the presence of some trajectories from the region of S going to the region of S' after some bounded time.

For illustrative purpose, consider the set of continuous variables X to be of dimension two and let $p(x, y)$ be a formula for which y is a function of a real variable x. The example $p(x, y) = -x + y$ is shown in Figure 4.5. The normal

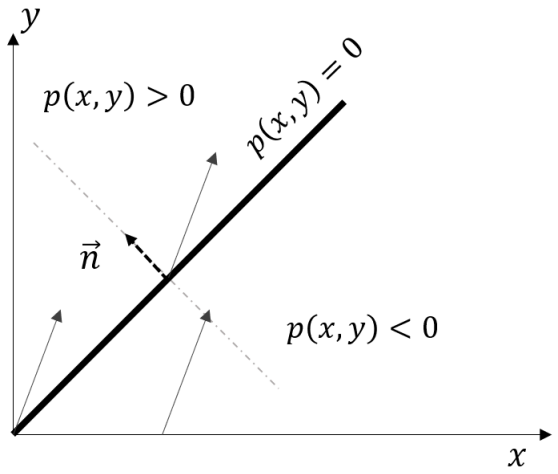


Figure 4.5: Projecting the flow onto the normal vector of qualitatively chosen boundaries

vector of $p(x, y)$ is obtained by applying the partial derivatives with respect to each variable. It is not other than the gradient of p, thus points towards the direction of increasing p when a small increase of x and y occurs, in other words points towards the half plane $p(x, y) > 0$.

$$\vec{n} = \left(\frac{\partial p}{\partial x}, \frac{\partial p}{\partial y}\right) = (-1, 1) \tag{4.3}$$

Let us consider an ODE represented on the figure by its vector field $F : \dot{x} = 1$, $\dot{y} = 2$. A sufficient condition for which a trajectory starting in $p(x, y) < 0$ is able to cross to $p(x, y) > 0$ is that there exists a point on the boundary $p(x, y) = 0$ for

which the vector field is pointing to $p(x, y) > 0$. We project the flow vector onto the normal vector and study the sign of the obtained scalar product $\vec{dp}.\vec{F}$. If there is a point (x_0, y_0) for which $\vec{dp}.\vec{F}(x_0, y_0) > 0$ then such a trajectory exists. In our example $\vec{dp}.\vec{F}(x, y) = 1$. If $\vec{dp}.\vec{F} = 0$ then for sure no trajectory from $p(x, y) < 0$ to $p(x, y) > 0$ is possible, but with this information alone we cannot guarantee that there is no transition from $p(x, y) < 0$ to $p(x, y) = 0$. In such case, we need to evaluate the vector field in a neighborhood around $p(x, y) = 0$ by studying the Jacobian matrix.

Regions defined using multiple constraints Consider our previous example and define another region boundary $p_2(x, y) = x + y$. We consider all the nine connected regions formed by $p_1(x, y)$ *op* $0 \wedge p_2(x, y)$ *op* 0 where *op* $\in \{<, >, =\}$. To evaluate whether or not transitions are possible from a region to another, we need to evaluate the scalar product over the boundary of each region. In our example, those boundaries for the four regions of dimension two are given by two half lines. E.g., for S_0 the boundary is $(p_1(x, y) = 0 \wedge x \leq 0) \wedge (p_2(x, y) = 0 \wedge x \geq 0)$ (see Figure 4.6).

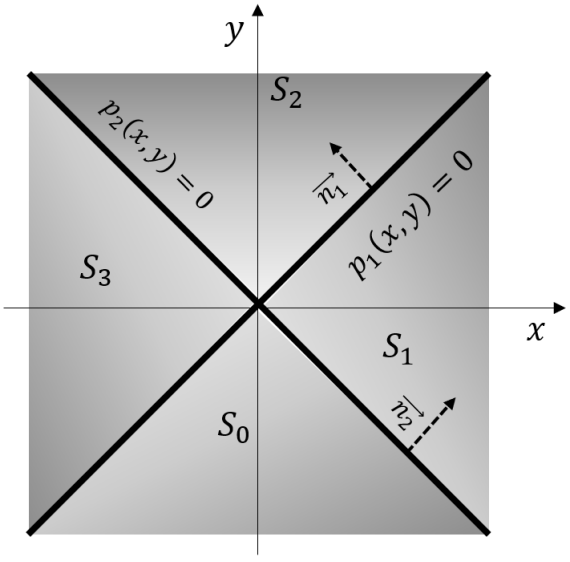


Figure 4.6: Flow projection onto regions defined using multiple constraints

Varying normal vector In the case of polynomial hybrid automata, the direction of the normal vector over a given boundary varies with the position on the boundary. In such case $\vec{dp}$ is expressed as a polynomial.

Evaluating the flow projection onto boundaries Consequently, to evaluate the presence or not of an abstract transition, one is required to find the truth values of semi-algebraic constraints. We use a method that has been elaborated tackling this problem called *Cylindrical Algebraic Decomposition (CAD)*. CAD was first introduced by Collins [56]. We now review briefly the basic steps of this method. Let $\{P(X)\}$ be a set of polynomials over n variables with coefficients in $\mathbb{Q}$. The method constructs a partition of $\mathbb{R}^n$ where the set of polynomials $\{P(X)\}$ holds a constant sign for all variables in X in every region of the computed partition. Once the decomposition is computed, it can be then used to give solutions to system of (in)-equations. The algorithmic steps are:

1. For each coordinate variable $x \in X$,
   (a) Find the projection on x of the intersection points of all polynomials in $P(X)$ (i.e., solve $p_1(x) = p_2(x) ... = p_n(x)$).
   (b) Find the zero sets of each polynomial with respect to x.
   (c) Find singularity points on x for every polynomial (i.e., the function does not behave well or is not defined).
   (d) Form regions on the domain of x partitioned according to the previously found sets of points and/or intervals.
2. Choose a sample point within each region, and evaluate each polynomial $p(x)$ at each chosen point for x and symbolically for the other coordinates (variables) from X.
3. Study the sign of each $p(x)$ at the sampled points.
4. Match the (in)-equalities required by the problem to be solved with the previously obtained table, if no match is found then the problem admits no feasible solution.

The CAD method has been proven to be doubly exponential [40]. However one can benefit from parallelism to alleviate the computation time, since every transition computation can be performed independently. The idea will be discussed in the next chapter.

Extending the expressiveness to rational fractions For the moment, the considered class of hybrid systems is the polynomial class. Nonetheless with some minor modifications, one can extend the abstract states and transitions computation to ODEs with rational functions. For example, given a rational fraction $\frac{P(x)}{Q(x)}$, Studying the truth value of $\frac{P(x)}{Q(x)} > 0$ is equivalent to studying the truth value of:

$$(P(x) > 0 \wedge Q(x) > 0) \vee (P(x) < 0 \wedge Q(x) < 0) \tag{4.4}$$

One can use CAD with an intermediary computation layer to allow for ODEs with rational fractions.

Example 4.5 (Transitions Computation)**.** Let us compute the transitions between abstract states from the previous example (Example 4.3, page 90). Consider the two adjacent regions given by abstract states S_8 and S_9 and their common boundary region given by S_{10} (Table.4.3, page 94).

$$S_8 : x_0^2 + x_1 > 0 \wedge x_0 + x_1^2 > 0 \wedge x_0^2 + x_1^2 > 4 \tag{4.5}$$

$$S_9 : x_0^2 + x_1 > 0 \wedge x_0 + x_1^2 > 0 \wedge x_0^2 + x_1^2 < 4 \tag{4.6}$$

$$S_{10} : x_0^2 + x_1 > 0 \wedge x_0 + x_1^2 > 0 \wedge x_0^2 + x_1^2 = 4 \tag{4.7}$$

The normal vector to the boundary region is given by $\vec{n} = (2x_0, 2x_1)$. The scalar product of the normal vector with the flow in mode q_0 is then:

$$\vec{n}.\vec{F} = 2x_0^3 + 2x_1^3 + 4x_0x_1 \tag{4.8}$$

The presence of a trajectory from S_9 to S_{10} and from S_{10} to S_8 is then given by the truth value of the following formula:

$$x_0^2+x_1^2 = 4 \wedge 2x_0^3+2x_1^3+4x_0x_1 > 0 \wedge x_0^2+x_1 > 0 \wedge x_0+x_1^2 > 0 \wedge x_0 \geq 0 \wedge x_1 \geq 0 \tag{4.9}$$

The evaluation of formula 4.9 using *CAD* returns *true*. The same reasoning goes for evaluating the presence of transitions from S_8 to S_{10} and S_{10} to S_9, the *CAD* evaluation of the corresponding formula gives *false*.

4.2.3 Linking the single mode abstractions

At this stage, the abstract states and transitions have been computed for one mode. The operation is repeated similarly for every mode q of H. The following operation will link the obtained mode abstractions. The linkage operation consists in expressing each guard region of a transition τ in a mode q_1 whose target mode is q_2 (or the image in q_2 of this region by the reset if the transition has a reset) according to the partition of mode q_2. In other words, we have to compute $\alpha_{d(q_2)}(G(\tau))$ (or $\alpha_{d(q_2)}(R(\tau)(G(\tau)))$ in case of reset), given $\alpha_{d(q_1)}(G(\tau))$, and the transitions from the regions of the latter one to the regions of the first one. The operation is implemented by checking the nonemptiness of $p_1 \cap G(\tau) \cap p_2 \cap Inv(q_2)$ (or $R(\tau)(p_1 \cap G(\tau)) \cap p_2 \cap Inv(q_2)$ in case of reset) for each region $p_1 \in d(q_1)$ and each region $p_2 \in d(q_2)$, which boils down to a satisfiability check of a system of polynomial equalities and inequalities. If satisfied, a mode change transition is thus added from region p_1 to region p_2.

4.2.4 Refinement operation.

Refining the abstraction consists in adding further information from the hybrid system to obtain a more precise abstraction. The idea is to split one or more regions into several regions by adding further boundaries to the decomposition. This is applied per mode of the hybrid system. What is important to notice is that many of the previously computed transitions remain valid after a refinement operation. The refinement algorithm checks which transitions remain valid and recomputes those that require to be reevaluated. The following are the algorithmic steps to perform for a refinement operation. Consider a new boundary region B_{new} (typically of dimension $n-1$) to be added to the abstraction. In order to keep the abstraction complete while incorporating the new boundary into the current partition, the following is performed:

1. Find the set of regions $\{R_{compute}\}$ from the previous decomposition that intersect B_{new}.
2. For the set of regions $\{R_{old}\}$ that do not intersect B_{new}, leave their incoming and outgoing transitions intact.

3. Process each region $R_{compute}$ intersecting B_{new}.

   - Find which boundaries (in all dimensions) of $R_{compute}$ intersect B_{new}.
   - Compute the newly obtained regions.
   - Recompute the in and out transitions relative to those new regions as previously explained in the subsection 4.2.2.

4.2.5 Computing time bounds

In this subsection, we are interested in algorithmic methods to compute the time bounds of the timed abstraction. The timeless abstraction $DH_{\mathfrak{P}}$ provides the sets of qualitatively chosen regions with transitions towards adjacent ones. Let P be a partition of $\mathbb{R}^n$ associated with one of the modes of H and $I_p = [t_{min}, t_{max}]$ the associated region time interval of some region $p \in P$. Finding the exact minimum and maximum sojourn times is not always possible. Hence, it is rather more practical to consider upper and lower bounds of these sojourn times that keep the abstraction complete.

Definition 4.15 (Reasonable time bounds)**.** *The upper and lower bounds t_{min}, t_{max} associated to some region p are said to be reasonable if $t_{min} > 0$ and $t_{max} \neq +\infty$, otherwise they are said unreasonable.*

Some interesting points to be noted when investigating whether or not a region has reasonable time bounds:

- An unbounded region does not necessarily have unreasonable time bounds.
- A bounded region with no critical points does not necessarily have reasonable time bounds.
- A region containing only critical points has unreasonable time bounds.
- A qualitatively partitioned region (according to the sign of the derivative vector) containing no critical points does not necessarily have reasonable time bounds.

Each boundary of a region p is associated with transitions towards adjacent regions in the abstract timeless system. The transition associated to a boundary is either (inward) or (outward) or (inward and outward) towards (or from) p. For a region expressed as a semi-algebraic set given by $p_1 < 0 \wedge p_2 < 0 \wedge ... \wedge p_n < 0$, the boundaries are $B(p) = \{p_1 = 0, ..., p_n = 0\}$. Let $B_{in}(p)$ be the set of inward boundaries towards p, $B_{out}(p)$ be the set of outward boundaries from p and $B_{in/out}(p)$ be the set of inward and outward boundaries towards (or from) p. The sign based timeless abstraction allows one to differentiate between regions containing only critical points and those that do not contain any critical point (the number of critical points in any region is finite as the number of zeros of a polynomial is finite). Regions containing only critical points are assigned a maximum and minimum sojourn time $t_{min} = t_{max} = +\infty$. Consequently, for the following the time bound computation concerns regions containing no critical points.

Assumption 4.2. *The considered region p contains no critical points.*

Numerical simulations can be used to find approximate time bounds of p, however this is not an exact method.

Numerical estimation of the time bounds We consider a set of points for each inward boundary in $B_{in}(p)$. A number of numerical simulations initialized from these points are computed and stopped when trajectories verify the equations of the abstract state neighboring any outward boundary of the region from $B_{out}(p)$. The exact trajectory is not guaranteed to reach an exit. Assuming the numerical trajectories reach an outward boundary, we obtain a collection of time lapses. Choosing the maximum and minimum elements of the sample allows one to determine approximations of the maximum and minimum sojourn times. Obviously, the obtained bounds are estimated and not verified, thus the timed abstraction obtained is not necessarily complete. We propose a more suitable and accurate method for computing the time bounds.

Guard satisfying regions If the considered region p (or part of it) satisfies a mode change condition (i.e., the guard condition) from mode q to mode q' while having the invariant of q satisfied within the region, the system exhibits non-determinism between either a mode jump or staying within the mode q. This fact must be reflected by the time constraint associated to p. The dynamics within the two modes q and q' must be taken into account for the region p and each associated region p' in q' reached after mode jump (possibly via a reset). The computation of the time bounds t_{min} (resp. t_{max}) should be done for both regions p and p' together as a whole by combining minimum (resp. maximum) sojourn times in p then p' with a non-deterministic mode change (and possible reset), i.e., evaluating the global minimum (resp. maximum) bounds for all possible instants of mode jump, by using both dynamics of q and q'. In the following, we consider the standard case of a region in absence of any guard or reset.

Assumption 4.3. *The considered region p does not intersect any guard or reset set.*

Computing t_{max}

We previously defined t_{max} as the maximum sojourn time in a region.

Consider a system of two-dimensional ODEs:

$$\dot{x} = f_1(x, y) \tag{4.10}$$

$$\dot{y} = f_2(x, y) \tag{4.11}$$

The problem is summarized as follows. Consider an initial and final point in the (x, y) plane $M_i(x_i, y_i)$ and $M_f(x_f, y_f)$. The time lapse $\tau = t_f - t_i$ taken by the continuous system initially in M_i at time t_i to reach M_f at time t_f is expressed by:

$$\tau = \int_{t_i}^{t_f} dt = \int_{x_i}^{x_f} \frac{dx}{f_1(x, y)} = \int_{y_i}^{y_f} \frac{dy}{f_2(x, y)} \tag{4.12}$$

Computing τ exactly is not always possible since, generally speaking, the solutions $x(t)$ and $y(t)$ cannot be expressed in closed forms. Suppose that M_i and M_f are varying points respectively on the inward boundary B_{in} and outward boundary B_{out}. To find a safe upper bound of the maximum sojourn time, we will use a

variant of flow-pipe construction. The idea is to construct a flow-pipe that will not be used to find the set of reachable states, but rather to identify the time elapsed to reach an outward boundary. The flow-pipe $Flow$ is of total duration $N\Delta t$ where Δt is the time step and $N \in \mathbb{N}$ is to be found. $Flow$ is initialized from an inward boundary $b_{in} \in B_{in}(p)$ of a considered region p. `ContinuousTimeElapsed`$(B, \Delta t)$ is a procedure computing a flow-pipe initially starting from B with a time step Δt that was introduced in the scientific context (Chapter 2, page 29). In the latter we reviewed the existing flow-pipe computation tools that were successful in analyzing hybrid automata from linear to polynomial dynamics [52, 27]. To the best of our knowledge, the tools assume convex initial sets, however the qualitatively chosen boundaries can be non-convex. Thus, practical results for time bound computation will only be applied to convex qualitative regions.

Algorithm 2 is applied to obtain t_{max} of a region p.

input : set $B_{in}(p)$ of inward boundaries of a region p, time horizon T
output: a safe over-approximating time bound t_{max} of p

for $b_{in} \in B_{in}(p)$ **do**
  $t_{max_i} \leftarrow 0$;
  while $t_{max_i} < T$ **do**
    $Flow \leftarrow$ `ContinuousTimeElapsed`$(b_{in}, \Delta t)$;
    $t_{max_i} \leftarrow t_{max_i} + \Delta t$;
    if $(Flow \cap p = \emptyset)$ **then** break;
  end
  if $(t_{max_i} \geq T)$ **then** return "max reached";
end
return `Maximum` (t_{max_i});

Algorithm 2: Computing a safe time bound of a region

Algorithm termination Remind that the region p is chosen qualitatively and not containing any critical point. This assumption does not prove that the time bounds are finite, but however discards some situations where the time bounds are not finite. Consequently the qualitative partition choice eases the time bounds computation. Obviously, unless bounded by T, termination of algorithm 2 is generally not guaranteed. One can check for conditions for which finite time bounds exist and adapt the maximum time horizon T accordingly.

Proposition 4.2 (Sojourn bounds)**.** *A sufficient but not necessary condition for the region p to have finite time bounds (t_{max} finite, thus nonnegative real) is that $\exists i\ 1 \leq i \leq n \quad \forall \boldsymbol{x} \in \overline{p} \quad \dot{\boldsymbol{x}}_i \neq 0$, where $\overline{p}$ is the closure of p.*

If the condition in proposition 4.2 is verified over p for a dimension i then all trajectories should respect finite time bounds for staying in the region p. On the other hand, a trajectory making a finite number of orbital spins once inside p then exiting p does not satisfy this condition while having finite time bounds. One way to look for a finite time bound is to refine the partition with the objective that the regions become small enough for the condition to hold for each of them. If the derivatives along each axis take each a finite number of null values inside the partition, but never all at the same time, there is no problem with refining the partition to have each null derivative point alone in one region. In the other cases, a further discussion on critical points is presented in Appendix B.
Note that the t_{max} computed by Algorithm 2 is only valid for those trajectories that from a $B_{in}(p)$ are guaranteed to exit by a $B_{out}(p)$. This happens because the flow-pipe is an over-approximation and some trajectories staying in p with $t_{max} = +\infty$ may exist. To deal with such trajectories, one can combine under and over-approximations to obtain faithful time bounds whether it is the maximum or minimum sojourn time.

Computing t_{min}

As previously defined, t_{min} corresponds to the minimum sojourn time in a state. Consequently, if any inward and outward boundaries of the region p have an intersection then we associate a null value to t_{min}. Thus, one can associate t_{min} to a specific entry boundary B_{in} instead of the whole region. A simple way to compute a lower bound of the time passed in p by all the trajectories, is to consider the minimal distance between B_{in} and B_{out} expressed as:

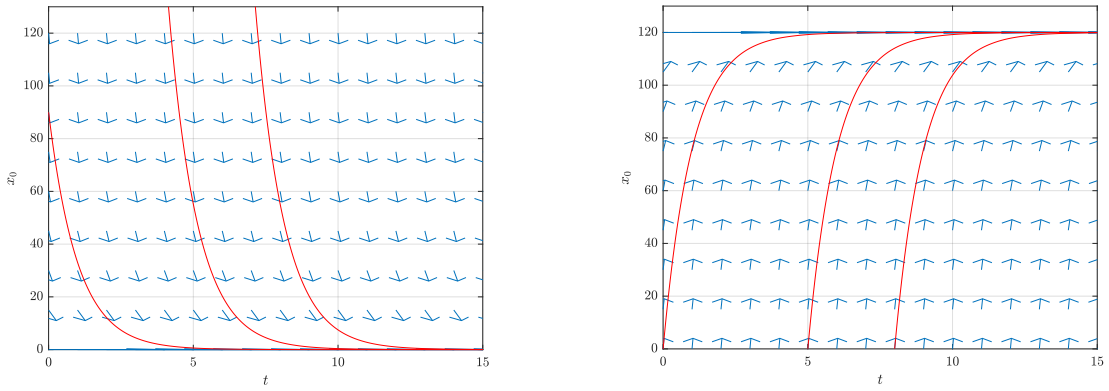
$$t_{min} = \frac{d_{min}}{v_{max}} \tag{4.13}$$

$$d_{min} = min_{M \in B_{in}, M' \in B_{out}} ||MM'|| \tag{4.14}$$

$$v_{max} = max_{x \in p} ||\dot{x}|| \tag{4.15}$$

For the class of polynomial hybrid automata we consider, solving the equations is a well-studied procedure in non-linear optimization with constraints.

Example 4.6 (Temperature regulator)**.** Consider again the temperature regulator, or thermostat (see Figure 3.1). The continuous behavior of each mode is illustrated in Figure 4.7.



(a) (b)

Figure 4.7: Thermostat phase plane with randomly generated numerical simulations: (a) Mode off, (b) Mode on

The previously explained abstraction algorithms are applied to this thermostat example. The result is automatically obtained from the tool (that will be presented in Chapter 5) and is illustrated in Figure 4.8. Green states correspond to guard satisfiability, blue states are regions interpreted from the other mode and purple states are regions produced if both mode invariants were set to *true*. The produced abstraction graph proves that the four regions $(off, 3)$, $(off, 4)$, $(on, 0)$ and $(on, 4)$ of the state space in purple color are time invariants. The abstract state of the initial set $[80, 90]$ is region 0 in mode off. From state 0 only one abstract trajectory exists, a mode change is possible to region 1 in mode on and going back to mode off is possible. The abstraction is sufficient to provide a proof that $(off, 2)$ and $(off, 1)$ are to be crossed by the system for a mode change to happen. The transitions outgoing from the blue regions are computed during the mode linkage step. Notice that the information within this abstraction are not enough to prove for example that all trajectories starting from $(off, 0)$ are

eventually (i.e., in finite time) subject to a mode change. This can be proven however if the abstraction contains additionally time sojourn bounds expressing how much time can the system stay in a given region as we previously defined (Def. 4.7).

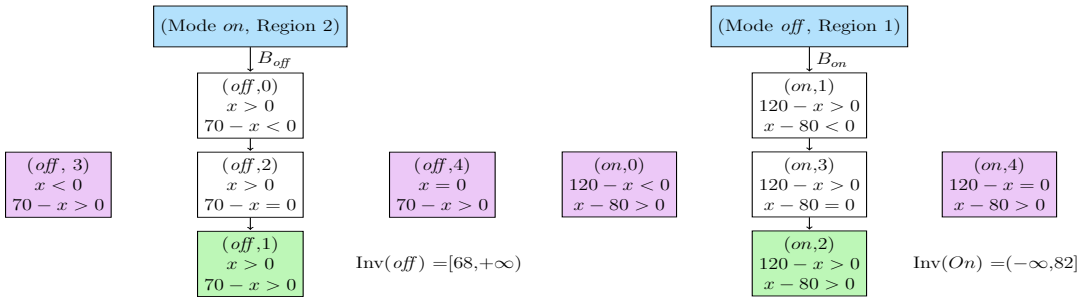


Figure 4.8: Thermostat qualitative abstraction: (a) Mode *off*, (b) Mode *on*

Attributing Sojourn Bounds Let us attribute sojourn bounds to the previous abstraction. The obtained timed abstraction, illustrated in Figure 4.9, is not computed presently by the tool, but the bounds are computed by hand by considering the slowest and fastest trajectories in each region. To illustrate the computation, we will consider the abstract state $(\mathit{off}, 0)$ and attribute the minimum and maximum sojourn time in the timed automaton.

Minimum sojourn time For the lower bound of the minimum sojourn time t_{min}, the minimum distance crossed by a trajectory under the continuity assumption is $d_{min} = 80 - 70 = 10$. Let us find the maximum (or an upper bound) of the first order derivative v_{max}. The dynamics of mode *off* is $\dot{x} = -x$, consequently the upper bound v_{max} is given by 80. The lower bound of the minimum sojourn time associated to the considered abstract state is thus:

$$t_{min} = \frac{d_{min}}{v_{max}} = \frac{10}{80} = 0.12$$

Maximum sojourn time Considering the same abstract state and taking into account that the exact solution to $\dot{x} = -x$ is $t = Ln(1/x)$, then an upper bound

for the sojourn time can be deduced by considering the trajectory initially at 80 and reaching 70:

$$t_{max} = ln\left(\frac{80}{70}\right) = 0.14$$

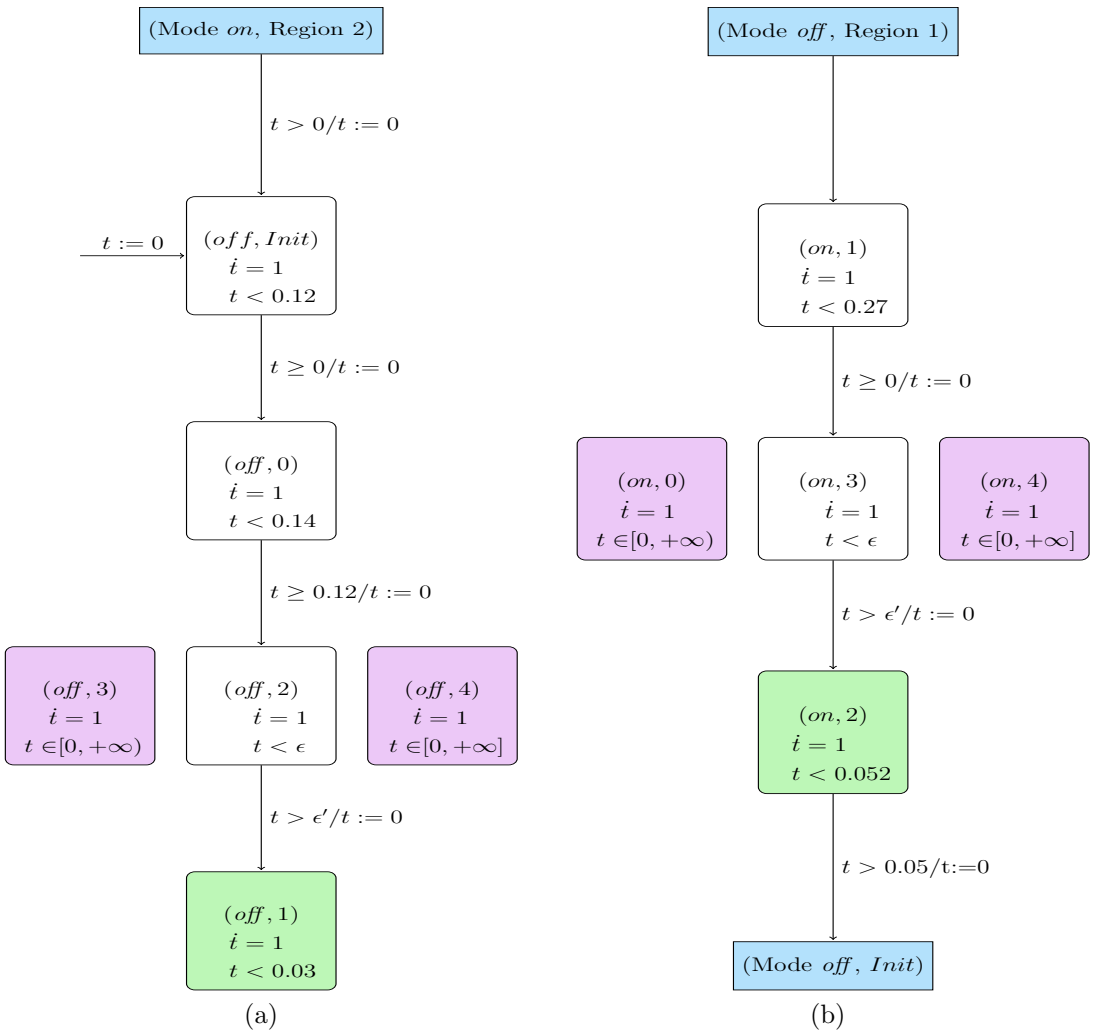


Figure 4.9: Timed abstraction of the thermostat system (a) Mode on, (b) Mode off

4.3 Application to diagnosability verification

Hybrid automata have generally an infinite state space and their semantics can be represented using infinite state machines. Thus, to verify formally their properties we will rest on using model checking over the abstraction of the hybrid system. Model-checking can be inconclusive, in which case the abstraction must be refined. In this section, we adapt counterexample guided abstraction refinement (CEGAR) that we previously reviewed (Chapter 2.3.5, page 40) to diagnosability verification. CEGAR was originally proposed to verify safety properties [48, 6].
Note that to verify safety properties, it is sufficient to check one execution at a time and verify whether the execution can reach an unsafe state. However, verifying diagnosability reveals a more complex task as one is required to simultaneously analyze two executions at a time, i.e., to verify whether or not the two executions have the same observations while only one of them contains the considered fault. The abstraction of the hybrid automaton can be directly used to verify safety and generate an abstract counterexample when safety is violated. This is performed by state exploration of the abstraction given the abstract states of the unsafe set of states. This is not the case for diagnosability verification, in fact a transformation of the abstraction is required to find critical pairs using state exploration.

4.3.1 CEGAR scheme for hybrid automata diagnosability verification

Consequently, the adaptation of CEGAR to verify diagnosability of H consists in three steps described as follows and to be detailed in the next subsections:

- **Abstracting a hybrid automaton** using the previously elaborated method to obtain the timed automaton abstracting H.
- **Diagnosability checking** of a timed abstract automaton of H. Unlike safety verification, the timed abstraction cannot be directly used to check for the property to be verified and generate abstract counterexamples. This is why we use the twin plant method that transforms the timed automaton

abstraction to another representation. The latter can be directly used to generate a counterexample $C.E$ by simple state exploration when diagnosability is not verified at the abstract level.

- **Validation of the *C.E*** by checking whether the $C.E$ is valid or spurious at the concrete level.
- **Refinement** of the timed abstract automaton by using a finer hybrid state space decomposition to avoid the found $C.E$ when it is declared spurious.

Verifying diagnosability of a hybrid automaton by checking it on its abstraction is justified because if the diagnosability property is verified for an abstraction, then it is verified also for the concrete hybrid system. This can be established by showing that a concrete counterexample of diagnosability lifts up into an abstract counterexample of diagnosability. Actually, given a hybrid automaton $H = (Q, X, S_0, \Sigma, F, Inv, T)$, two executions h, $h' \in [[H]]$ such that

$$h = (q_0, \mathbf{x}_0) \xrightarrow{l_0} (q_1, \mathbf{x}_1)...(q_i, \mathbf{x}_i) \xrightarrow{l_i} ...$$
$$h' = (q'_0, \mathbf{x}'_0) \xrightarrow{l'_0} (q'_1, \mathbf{x}'_1)...(q'_i, \mathbf{x}'_i) \xrightarrow{l'_i} ...$$

are called a counterexample of diagnosability in H with respect to the fault F if they satisfy the three conditions defined in Definition 3.16 (page 71), i.e., if h and h' constitute a critical pair of H. We will denote each state $(q_i, \mathbf{x}_i)$ by s_i and $(q'_i, \mathbf{x}'_i)$ by s'_i.

Theorem 4.3. *Given a hybrid automaton H, a timed abstract automaton $TH_{\mathfrak{P}}$ of H with abstraction function $\alpha^t_{\mathfrak{P}}$ and a modeled fault F in H, if F is (Δ-)diagnosable in $TH_{\mathfrak{P}}$ then F is (Δ-)diagnosable in H.*

Proof. Assume that F is not Δ-diagnosable in H. Then a critical pair exists among the executions of H, thus the following holds:

$$\exists h, h' \in [[H]], F \in h \text{ and } F \notin h' \text{ and } time(h, F) \geq \Delta \text{ and } Obs(h') = Obs(h).$$

From Theorem 4.2, since $TH_{\mathfrak{P}}$ is a timed abstraction of H with abstraction function $\alpha^t_{\mathfrak{P}}$, if we denote $\hat{h} = \alpha^t_{\mathfrak{P}}(h)$ and $\hat{h}' = \alpha^t_{\mathfrak{P}}(h')$, then the following holds:

$$\exists \hat{h}, \hat{h}' \in [[TH_{\mathfrak{P}}]], F \in \hat{h} \text{ and } F \notin \hat{h}' \text{ and } time(\hat{h}, F) \geq \Delta \text{ and } Obs(\hat{h}') = Obs(\hat{h}).$$

Consequently, $(\hat{h}, \hat{h}')$ is a critical pair of $TH_{\mathfrak{P}}$, which establishes thus a counterexample of diagnosability in $TH_{\mathfrak{P}}$: $C.E = (\hat{h}, \hat{h}')$. By contradiction, this proves the result. □

Algorithm 3 illustrates the CEGAR scheme adaptation for hybrid automata diagnosability verification.

input : Hybrid Automaton := H; Considered Fault := F ; Precision := (integer, maximum number of refinements) *precision*
output: Decision := H is diagnosable | H is not diagnosable | Precision is reached

$TH \leftarrow$ Initial Timed Abstract Automaton of H;
$C.E \leftarrow$ Diagnosability Check (TH, F);
$abstraction\_level \leftarrow 0$;
while $C.E \neq \emptyset \wedge abstraction\_level < precision$ **do**
  if *Validate*$(C.E, H)$ **then** $decision \leftarrow false$;
  $Exit$;
  else $TH \leftarrow$ Refine $(TH, C.E, H)$;
  $C.E \leftarrow$ Diagnosability Check (TH, F);
  $abstraction\_level \leftarrow abstraction\_level + 1$;
end
if $C.E = \emptyset$ **then** $decision \leftarrow true$;
else $decision \leftarrow max\_reached$;

Algorithm 3: CEGAR scheme for hybrid automata diagnosability verification

4.3.2 Twin plant based diagnosability checking

Diagnosability checking of a discrete event system, modeled as an automaton, based on the twin plant method [67, 108] is polynomial in the number of states (actually it has been proved it is NLOGSPACE-complete [94]). The idea is to construct a non-deterministic automaton, called pre-diagnoser or verifier, that preserves only all observable information and appends to every state the knowledge about past fault occurrence. The twin plant is then obtained by synchronizing the pre-diagnoser with itself based on observable events to get as paths in the twin plant all pairs of executions with the same observations in the original system. Each state of the twin plant is a pair of pre-diagnoser states that provide two

possible diagnoses. A twin plant state is called an ambiguous one if the corresponding two pre-diagnoser states give two different diagnoses (presence for one and absence for the other of a past fault occurrence). A *critical path* is a path in the twin plant with at least one ambiguous state cycle. It corresponds to a critical pair and it has thus been proved that the existence of a critical path is equivalent to non-diagnosability. The twin plant method has been adapted to be applied to timed automata [105], where a twin plant is constructed in a similar way except that the time constraints of two executions are explicitly taken into account using clock variables. The idea is to verify whether the time constraints can further distinguish two executions by comparing the occurrence time of observable events. The definition of a critical path in the twin plant is analog, except that ambiguous state cycle is replaced by infinite time ambiguous path.

Lemma 4.1. *A fault is diagnosable in a timed automaton iff its twin plant contains no critical path [105].*

For timed automata, checking diagnosability is PSPACE-complete.

Algorithmic Computation of the Twin Plant of a Timed Automaton We review (informally) the main steps for building the twin plant of a partially observable timed automaton with a single modeled fault.

1. Make two identical copies of the timed automaton.
2. For one copy, rename the unobservable events (faulty and normal events), discrete states and clocks such that it has different names than the other copy.
3. Remove for one of the copies all transitions labeled with the copy's faulty event.
4. Compute the product of the two timed automata by forcing observable transition events to synchronize.
5. Assign the invariant of every state from the product timed automaton by composing the invariants of the merged states with the Boolean $\wedge$ operator.

Example 4.7 (Twin plant construction example)**.** Figure 4.10 illustrates a timed automaton with initial state q_{init} and observable events $\Sigma_o = \{a, b\}$ and unobservable event $\Sigma_u = \{u\}$ and fault event $\Sigma_f = \{f\}$. Under any two nominal and faulty executions of the timed automaton, the event b cannot appear at the same time. Consequently, the timed automaton is diagnosable and contains no critical pair. Notice that the invariants in q_1 and q_1^F play a role in determining diagnosability as b becomes an event forced to appear at necessarily different times between a non-faulty and a faulty execution. If the invariant of q_1 was for example the same as q_1^F, then the timed automaton would become non-diagnosable. If we change the guard of the transition (q_0, f, q_1^F) from $t > 3$ to $t \geq 3$ then the timed automaton becomes non-diagnosable witnessed by the critical pair (h, h') such that:

$$h = (q_{init}, 0) \xrightarrow{a} (q_0, 0) \xrightarrow{3} (q_0, 3) \xrightarrow{u} (q_1, 3) \xrightarrow{b} (q_2, 3)$$
$$h' = (q_{init}, 0) \xrightarrow{a} (q_0, 0) \xrightarrow{3} (q_0, 3) \xrightarrow{f} (q_1^F, 3) \xrightarrow{b} (q_2^F, 3)$$

Projecting the pair onto observations we obtain $Obs(h) = Obs(h') = 0, a, 3, b$.

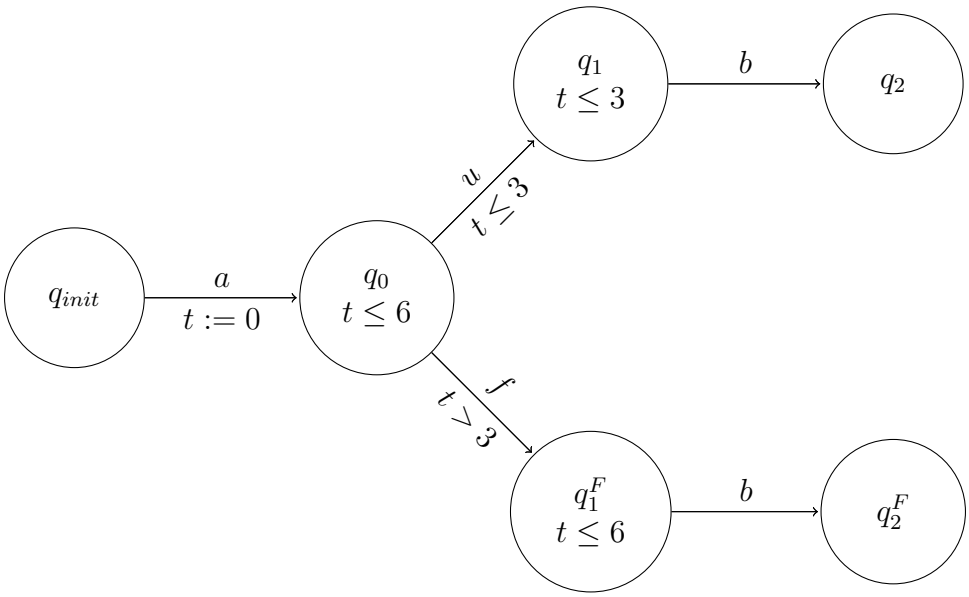


Figure 4.10: Diagnosable Timed Automaton

The twin plant is the product, synchronized on observable events, of the timed automaton with a copy of itself where the copy holds only the normal executions

(those that do not contain the fault event). For the previous example, every state q is copied and denoted q', unobservable event u is copied and renamed u' and clock t is copied and renamed t'. The observable events a and b are synchronized, while transitions of u and u' can occur asynchronously. The obtained twin plant is illustrated in Figure 4.11. Using a simple state exploration algorithm of the twin plant, we cannot find any fault labeled execution with infinite time proving the absence of critical pairs in the original timed automaton, and thus the diagnosability.

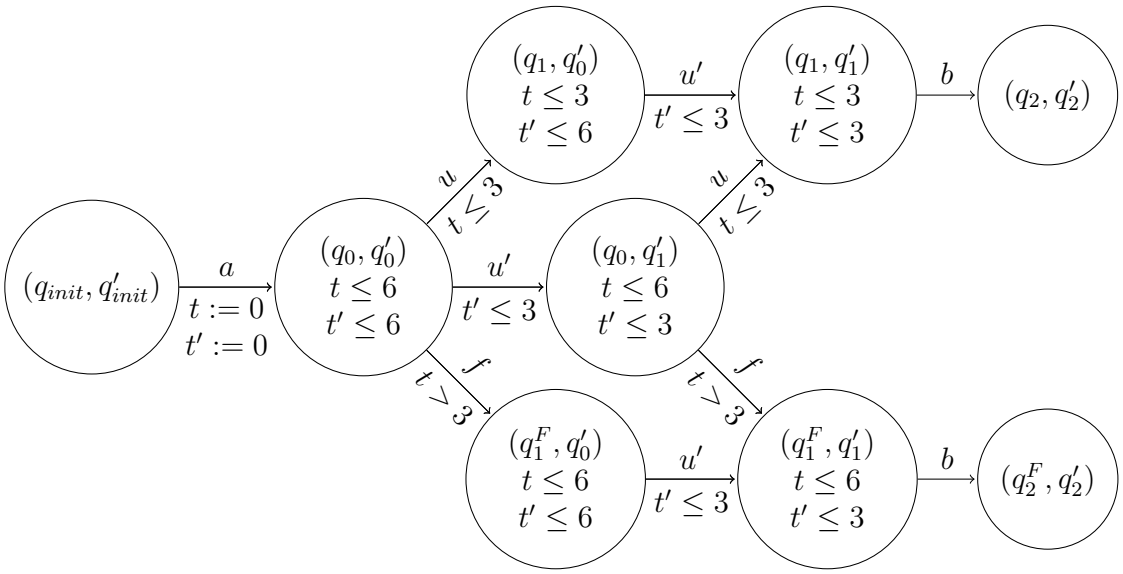


Figure 4.11: Twin plant of a diagnosable timed automaton

4.3.3 Counterexample validation or refusal

After applying the twin plant method on a timed abstract automaton $TH_{\mathfrak{P}}$ of H as described in [105], suppose that a critical pair $C.E = (\hat{h}, \hat{h}')$ is returned (if not, it means that $TH_{\mathfrak{P}}$, and thus H, is diagnosable). Whether we find or not two concrete executions $h, h' \in [[H]]$ whose abstractions by $\alpha^t_{\mathfrak{P}}$ are $\hat{h}$ and $\hat{h}'$ and form a concrete critical pair decides if $C.E$ is validated or refuted. We detail below both procedures for validation or refusal and the reasons for which, in the latter case, a critical pair can be assumed spurious.

Validating an abstract counterexample: If the returned counterexample $(\hat{h}, \hat{h}')$ exists at the concrete level, there exists $h, h' \in [[H]]$, whose abstractions

by $\alpha^t_{\mathfrak{P}}$ (according to Theorem 4.2) are $\hat{h}, \hat{h'}$, and which form a critical pair. This concrete pair is a witness of non-diagnosability of the hybrid automaton H.

In practice, validating the counterexample involves computing reachable sets of states. As we have seen, computing the exact set is not possible, one uses computations of safe over-approximations instead, such as using ellipsoids and zonotopes for complex dynamics or hyper-cubes for simpler ones [3, 19]. Two flow-pipes are computed with pre-defined time horizons T and T' such that $T = time(\hat{h})$ and $T' = time(\hat{h'})$. The flow-pipes trace what is dictated by the counterexample. Suppose that $\hat{h'}$ is the fault carrying execution of the counterexample, then the verification of Δ-diagnosability will translate by the flow-pipe tracing $\hat{h'}$ to be of duration at least Δ in the faulty mode.

If the counterexample is not validated, the abstract counterexample $C.E$ is said spurious and becomes refuted.

Refuted counterexample: In case of spurious $C.E = (\hat{h}, \hat{h'})$, the idea is to construct longest finite executions $h, h' \in [[H]]$, that abstract by $\alpha^t_{\mathfrak{P}}$ into finite prefixes of $\hat{h}, \hat{h'}$ and such that $Obs(h) = Obs(h')$. The fact they cannot be extended means that $\forall \overline{h} \in [[H]]/h$, $\overline{h'} \in [[H]]/h'$ one step executions, either (i) $\hat{s}_{|h|+1} \neq \alpha^t_{\mathfrak{P}}(s_{|h|+1})$ (or $\hat{s}'_{|h'|+1} \neq \alpha^t_{\mathfrak{P}}(s'_{|h'|+1})$) or (ii) $\hat{l}_{|h|} \neq l_{|h|}$ (or $\hat{l}'_{|h|} \neq l'_{|h|}$) or (iii) $Obs(\overline{h}) \neq Obs(\overline{h'})$. In this case, $s^{reach}_{|h|+1}$ and $s'^{reach}_{|h'|+1}$ are returned, that represent the two sets of reachable concrete states that are the first ones to disagree with the abstract $C.E$. We summarize below the reasons resulting in the $C.E$ being spurious.

Spurious state reachability: There is no concrete execution in H whose abstraction is one of $\hat{h}$ or $\hat{h'}$, as one of the set of states of H whose abstraction is an abstract state $\hat{s}_i$ or $\hat{s}'_i$ is not reachable in H starting from the initial states of H. Note that care will have to be taken when refining $TH_{\mathfrak{P}}$ (see next subsection). E.g., a possible case is that there exists two executions (h, h') reaching (s_1, s'_1) and then (s_2, s'_2) but not reaching (s_3, s'_3) (and none passing by (s_1, s'_1) and (s_2, s'_2) reaches (s_3, s'_3)), and two other executions (u, u') reaching (s_2, s'_2) from $(s, s') \neq (s_1, s'_1)$, with $\alpha^t_{\mathfrak{P}}(s) = \alpha^t_{\mathfrak{P}}(s_1)$ and $\alpha^t_{\mathfrak{P}}(s') = \alpha^t_{\mathfrak{P}}(s'_1)$, and then reaching (s_3, s'_3), all with time periods compatible to those of the abstract executions. If the refined model simply eliminated the transition from $\hat{s}_2$ to $\hat{s}_3$ or from $\hat{s}'_2$ to $\hat{s}'_3$ then it could no longer be considered an abstraction of H, since some concrete execution in H would have

no abstract counterpart. Thus the refinement has to apply the split operation as previously described, so that preserving the abstraction while eliminating the spurious counterexample.

Spurious time constraints satisfaction: The abstract critical pair, when considered timeless, owns a concrete critical pair realization in H but none verifying the time bounds imposed by the abstract timed automaton. In this case it is not a spurious state reachability problem but a spurious timed state reachability problem. Actually the time constraints of the abstract critical pair cannot be satisfied by any concrete critical pair realizing it in H.

Spurious observation undistinguishability: The two executions of the abstract critical pair share the same observations (observable events with their occurrence times and snapshots of the values of observable continuous variables at arrival times in each abstract state) but actually any two concrete executions realizing this critical pair in H do distinguish themselves by the observation of some observable continuous variable.

4.3.4 Refinement of the abstraction

If it reveals that the abstract counterexample $C.E = (\hat{h}, \hat{h}')$ is spurious, then one refines the timed abstract automaton $TH_{\mathfrak{P}}$ to get $TH_{\mathfrak{P}'}$, guided by the information from $C.E$. The first step is analyzing $C.E$ to identify the reasons why it is spurious (as classified previously). The idea is to avoid getting relatively close spurious abstract counterexample when applying twin plant method on the refined timed abstract automaton $TH_{\mathfrak{P}'}$. The refinement procedure is described as follows and will be illustrated on our example in the next subsection.

1. Suppose that $C.E$ is refuted due to an illegal stay, i.e., the corresponding invariant is not respected. The consequence could be $s^{reach}_{|h|+1} = \emptyset$, i.e., an illegal transition. To eliminate such spurious counterexample next time, one can partition the region containing $\hat{s}_{|h|}$ to get a new region representing the legal stay such that the refinement can be done based on this partition. The idea is to eliminate illegal (unobservable) transitions between the new region and others by tightening time constraints. In a similar way, one can handle spurious counterexamples with illegal transitions due to the unsatisfiability of the corresponding guards by the

evolution of continuous variables, but with a legal stay this time.
2. Suppose that the refutation of $C.E$ is due to different observations from $s_{|h|+1}$ and $s'_{|h'|+1}$ without reachability problem. The idea is to calculate the exact moment, denoted $t_{spurious}$, before which it is still possible to get the same observations while after it the observations will diverge. With $t_{spurious}$, one can partition $\hat{s}_{|h|+1}$ and $\hat{s}'_{|h'|+1}$ to get a new region whose legal stay is limited by $t_{spurious}$ and transition to another region gives birth to a new refined observation by means of an observable continuous variable if any.

4.3.5 Case study example

The CEGAR scheme for diagnosability checking of hybrid automata will be illustrated by the following case study example.

Example 4.8 (Fault tolerant thermostat model)**.** We start from our thermostat example as nominal behavior and we add two faulty transitions to new faulty modes. The two observable events B_{on} and B_{off} allow one to witness mode changes and the continuous variable x is assumed to be observable. The system starts from $x \in [80, 90]$. Two faults are modeled as unobservable events F_1 and $F_2 \in \Sigma_f$ shown in Figure 4.12. In practice, the fault F_1 models a bad calibration of the temperature sensor. As for fault F_2, it represents a defect in the heater impacting its efficiency and is modeled by a parametric change of a constant in the expression of the first order derivative of x.

CEGAR scheme for fault F_1

Initial Abstraction: We consider an initial decomposition $\mathfrak{P} = \{P_{off}, P_{on}, P_{off}^{F_1}, P_{on}^{F_1}$ $P_{off}^{F_2}$, $P_{on}^{F_2}\}$ of the hybrid state space. Each partition $P \in \mathfrak{P}$ is made up of only one region representing the reals $\mathbb{R}$. Hence computing t_{min} and t_{max} for each region p yields $I_p = [0, +\infty)$, in other words the initial abstraction contains no time constraints.

Diagnosability Check: The diagnosability check using the twin plant method gen-

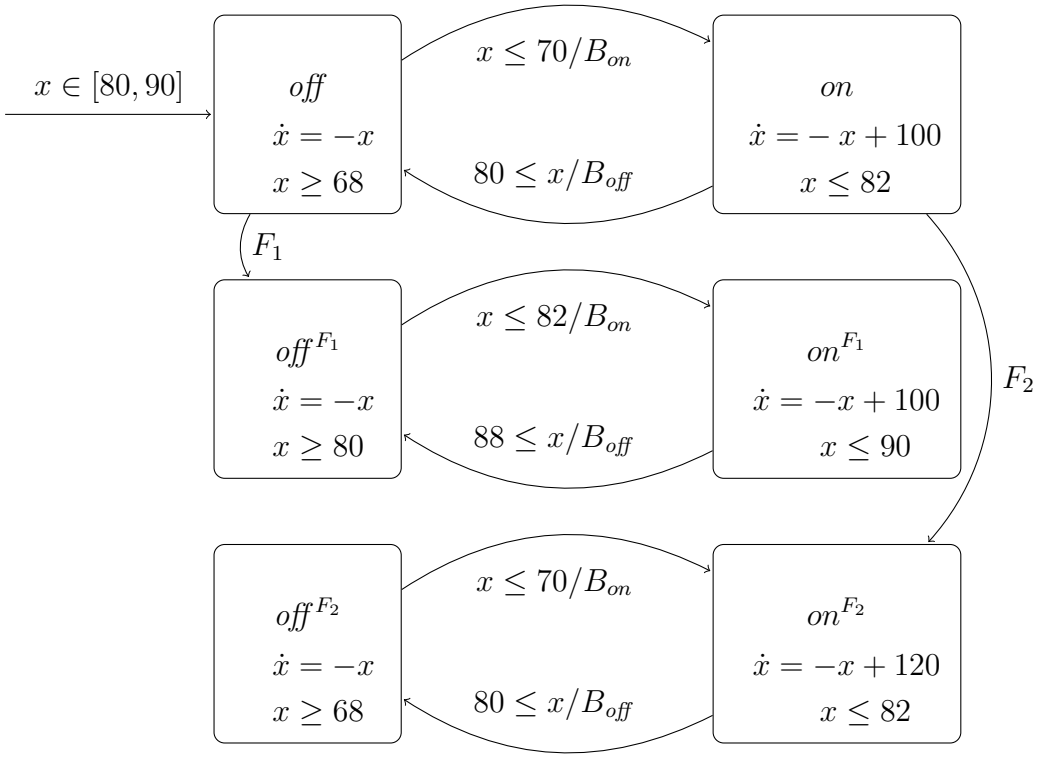


Figure 4.12: 1-dimensional hybrid automaton modeling a faulty thermostat

erates a counterexample $C.E = (\hat{h}, \hat{h'})$ such that:

$\hat{h} = (\mathit{off}, p_{\mathit{off}}) \xrightarrow{0.15} (\mathit{off}, p_{\mathit{off}}) \xrightarrow{Bon} (on, p_{on}) \xrightarrow{0.3} (on, p_{on}) \xrightarrow{B_{\mathit{off}}} (\mathit{off}, p_{\mathit{off}})...$
$\hat{h'} = (\mathit{off}, p_{\mathit{off}}) \xrightarrow{0.1} (\mathit{off}, p_{\mathit{off}}) \xrightarrow{F_1} (\mathit{off}^{F_1}, p_{\mathit{off}}^{F_1}) \xrightarrow{0.05} (\mathit{off}^{F_1}, p_{\mathit{off}}^{F_1}) \xrightarrow{B_{on}} (on^{F_1}, p_{on}^{F_1}) \xrightarrow{0.3}$
$(on^{F_1}, p_{on}^{F_1}) \xrightarrow{B_{\mathit{off}}} (\mathit{off}^{F_1}, p_{\mathit{off}}^{F_1})...$

Validation or refusal: The computation of the set of concrete executions $\{h\}$ (resp. $\{h'\}$) whose abstraction is $\hat{h}$ (resp. $\hat{h'}$) yields an approximation as follows:

$\{h\} = (\mathit{off}, [80, 90]) \xrightarrow{0.15} (\mathit{off}, [69, 77]) \xrightarrow{Bon} (on, [69, 70])...$
$\{h'\} = (\mathit{off}, [80, 90]) \xrightarrow{0.1} (\mathit{off}, [72, 81]) \xrightarrow{F_1} (\mathit{off}^{F_1}, [80, 81]) \xrightarrow{0.05}$
$(\mathit{off}^{F_1}, [76, 77] : \mathit{invalid})$

The concrete state computations show that it is not possible to stay 0.05 time units in mode off^{F_1} as the temperature reached would be [76,77] violating the invariant $x \geq 80$. The $C.E$ is thus refuted.
Refinement of the state space: The refinement aims at eliminating the previous spurious $C.E$. From this C.E, it is possible to compute the exact time constraint for staying in mode off^{F_1} and then triggering the transition B_{on} and refine the hybrid state space accordingly. Once refined, the new abstraction should not contain similar counterexamples. The validation process reveals that all trajectories entering mode off^{F_1} with $x \in [80, 81]$ cannot stay more than $t_{max} = 0.0124$ time units but it is possible for some trajectories to instantaneously change from off to off^{F_1} to on^{F_1} in which case $t_{min} = 0$, thus $I_{(\mathit{off}^{F_1}, [80,81])} = [0, 0.0124]$. The refined abstraction would carry this new information by updating the partition of mode off^{F_1}, from $\mathbb{R}$ to $(-\infty, 80) \uplus [80, 81] \uplus (81, +\infty)$, thus ensuring that all future generated counterexamples would satisfy this constraint.

CEGAR scheme for fault F_2
Initial Abstraction: The same as for F_1.
Diagnosability Check: The diagnosability check of the initial abstraction using the

twin plant method generates a $C.E = (\hat{h}, \hat{h'})$:

$$\hat{h} = (\mathit{off}, p_{\mathit{off}}) \xrightarrow{0.5} (\mathit{off}, p_{\mathit{off}}) \xrightarrow{B_{on}} (on, p_{on}) \xrightarrow{0.5} (on, p_{on}) \xrightarrow{B_{\mathit{off}}} (\mathit{off}, p_{\mathit{off}})...$$
$$\hat{h'} = (\mathit{off}, p_{\mathit{off}}) \xrightarrow{0.5} (\mathit{off}, p_{\mathit{off}}) \xrightarrow{B_{on}} (on, p_{on}) \xrightarrow{0.4} (on, p_{on}) \xrightarrow{F_2} (on^{F_2}, p_{on}^{F_2}) \xrightarrow{0.1}$$
$$(on^{F_2}, p_{on}^{F_2}) \xrightarrow{B_{\mathit{off}}} (\mathit{off}^{F_2}, p_{\mathit{off}}^{F_2})...$$

Validation or refusal: The computation of the set of concrete executions $\{h\}$ (resp. $\{h'\}$) whose abstraction is $\hat{h}$ (resp. $\hat{h'}$) yields an approximation as follows:

$$\{h\} = (\mathit{off}, [80, 90]) \xrightarrow{0.5} (\mathit{off}, [48.5, 54.58] : \mathit{invalid})$$
$$\{h'\} = (\mathit{off}, [80, 90]) \xrightarrow{0.5} (\mathit{off}, [48.5, 54.58] : \mathit{invalid})$$

This C.E is refuted due to illegal stay in the mode *off* violating the corresponding invariant. In other words the trajectories are not feasible: if the system stays in mode *off* for 0.5 time units then the state invariant is no longer true. Thus, if B_{on} is observed then the duration of stay in *off* should be smaller.

Refinement of the state space: To prevent future similar spurious counterexamples, a refinement is applied to the initial abstraction. The refined model considers new regions in mode *off*: $p_{\mathit{off}_1} = [80, 90]$ (initial region) and $p_{\mathit{off}_2} = [68, 80)$ (legal region). The computation of the time intervals relative to each region are: $I_{(\mathit{off},[68,80))} = [0, 0.16]$ and $I_{(\mathit{off},[80,90])} = [0, 0.12]$. The refined abstraction will encode these time constraints and ensure that a set of similar counterexamples (including this one) are eliminated. Regions that are not reachable will be eliminated, such as $[0, 68)$.

Second Abstraction

Diagnosability Check: The second $C.E$ generated from the refined twin plant is:

$$\hat{h} = (\mathit{off}, p_{\mathit{off}_1}) \xrightarrow{0.08} (\mathit{off}, p_{\mathit{off}_1}) \xrightarrow{\epsilon} (\mathit{off}, p_{\mathit{off}_2}) \xrightarrow{0.07} (\mathit{off}, p_{\mathit{off}_2}) \xrightarrow{B_{on}} (on, p_{on}) \xrightarrow{0.5}$$
$$(on, p_{on}) \xrightarrow{B_{\mathit{off}}} (\mathit{off}, p_{\mathit{off}})...$$
$$\hat{h'} = (\mathit{off}, p_{\mathit{off}_1}) \xrightarrow{0.08} (\mathit{off}, p_{\mathit{off}_1}) \xrightarrow{\epsilon} (\mathit{off}, p_{\mathit{off}_2}) \xrightarrow{0.07} (\mathit{off}, p_{\mathit{off}_2}) \xrightarrow{B_{on}} (on, p_{on}) \xrightarrow{0.4}$$
$$(on, p_{on}) \xrightarrow{F_2} (on^{F_2}, p_{on}^{F_2}) \xrightarrow{0.1} (on^{F_2}, p_{on}^{F_2}) \xrightarrow{B_{\mathit{off}}} (\mathit{off}^{F_2}, p_{\mathit{off}}^{F_2})...$$

Validation or refusal: Note that the continuous transitions in the second $C.E$ respect the temporal constraints added during the refinement based on the first

$C.E$. The corresponding concrete approximate executions of this $C.E$ are:

$\{h\} = (off, [80, 90]) \xrightarrow{0.15} (off, [69, 77]) \xrightarrow{B_{on}} (on, [69, 70]) \xrightarrow{0.5} (on, [80.5, 81.8])...$
$\{h'\} = (off, [80, 90]) \xrightarrow{0.15} (off, [69, 77]) \xrightarrow{B_{on}} (on, [69, 70]) \xrightarrow{0.4} (on, [79, 79.9]) \xrightarrow{F_2}$
$(on^{F_2}, [79, 79.9]) \xrightarrow{0.1} (on^{F_2}, [82.9, 83.7])...$

In this case, this second $C.E$ is also considered as spurious because, given the time constraints, the two trajectories are different in the observations of x in the hybrid system since the last regions are disjoint, i.e., $[80.5, 81.8] \cap [82.9, 83.7] = \emptyset$. And also the invariant $x \leq 82$ in on^{F_2} is violated.

Refinement of the state space: The counterexample analysis could identify the time boundary $t_{spurious}$, up to which the observations could be the same for at least two concrete trajectories, and after which the critical pair becomes spurious. In our example, suppose the fault occurred at t_f where $x \in [a, b]$, then $t_{spurious}$ is the time instant from which faulty and nominal sets of trajectories are disjoint:

$$t_{spurious} = ln\left(\frac{b - a + 20}{20}\right) + t_f \tag{4.16}$$

For the second spurious $C.E$, $t_f = 0.4$ and $t_{spurious} - t_f = 0.044$. The two concrete nominal and faulty executions originating from $(on, [79, 79.9])$ will be in the following temperature range after 0.044 time units: $x \in [79.90, 80.7]$ in the mode on and $x \in [80.7, 81.6]$ in the mode on^{F_2}. Hence, at any future time, the observations are different. By incorporating the time constraint in the refined abstraction, we ensure that counterexamples that are spurious because of disjoint observations including the previous one cannot be generated again. For the sake of simplicity, we analyzed the two faults separately. One more sophisticated strategy is to analyze the next fault based on the refined abstraction obtained from the analysis of the precedent fault.

4.4 Chapter Summary

In this chapter we elaborated a method to generate an abstraction of a given hybrid automaton using qualitative reasoning. A timing of the abstraction is proposed using existing flow-pipe techniques to generate the timed automaton abstraction. An

adaptation of CEGAR that was specifically used for safety verification is applied to diagnosability verification by extending results that were specific for discrete event and timed systems.

Chapter 5

Implementation and Experimental results

5.1 Automating the abstraction computation

In this chapter an implementation of a tool that applies previous algorithms from Chapter 4, Section 4.2 (page. 89) is presented. The tool automatically computes the timeless abstraction of a hybrid automaton and is implemented in `C++`. First, the tool architecture is presented followed by the primary classes and external dependencies. Second, we briefly explain the implementation specific procedures and present experimental results while evaluating performances. Finally, perspectives to extend the current implementation to compute the timed abstraction and perform diagnosability verification are presented and discussed.

5.1.1 Black box view

The input model is a polynomial hybrid automaton written in accordance to the grammar rules described in Appendix A.1. The tool takes the file $[name].model$ and performs the abstraction procedure if no model parsing errors are detected. The input file contains the different components of a hybrid automaton model H:

- A list of hybrid automaton modes
- A list of ODE's relative to each mode

- Invariants for each mode (if not specified then the invariant is set to true by default)
- Mode change transitions with their guards and resets
- Refinement functions
- Optionally an initial set

5.1.2 Tool Architecture

The tool architecture is presented in Figure 5.1. Execution flow shows how the applied algorithms are sequenced. The required libraries for each step of the computation are also mentioned. First, a parsing operation is applied to a given input model satisfying the previously presented grammar. Second, the state partitioning algorithm is applied to discretize the state space into qualitative regions. Then transitions are evaluated from every qualitative region with its adjacent neighbors. The obtained partitioning and transition evaluation results are then saved graphically as a finite state machine graph. At this point of the execution, the user can choose or not to compute the maximum and minimum first order derivatives of $\dot{X}$ in each partitioned region. The latter can in some situations be used to deduce the minimum and maximum sojourn times combined with the crossing distances of each region.
The following freely available libraries and solvers are required by the tool:

- `QepCAD` a Quantifier Elimination applying Cylindrical Algebraic Decomposition solver;
- `Nlopt` a non linear optimization library;
- `Boost` libraries for parsing inputs and generating outputs using regular expressions;
- `Graphviz` and `Matplotlib` for plotting and visualization.

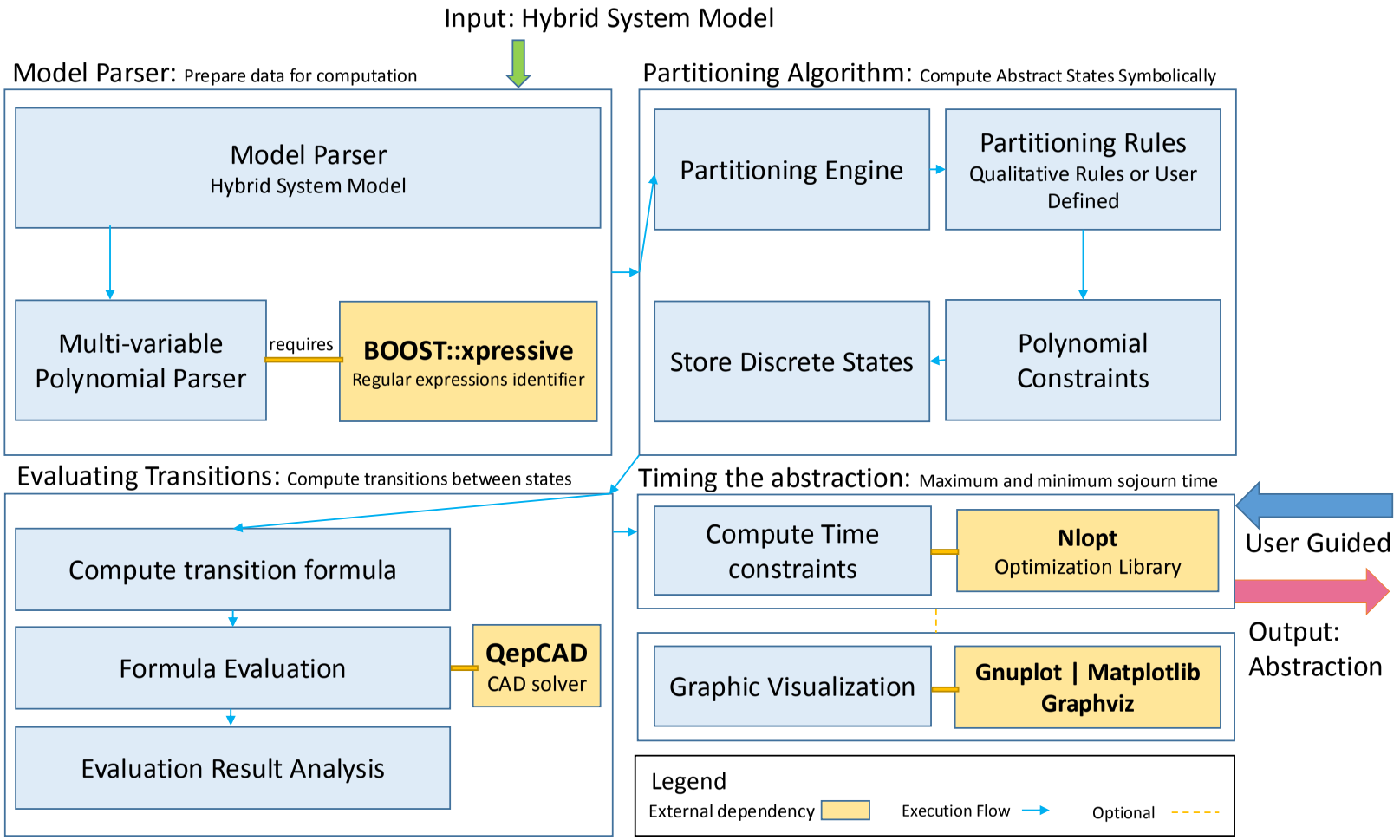


Figure 5.1: Architecture of the Hybrid Automata Qualitative Abstraction Tool

5.1.3 Classes and Methods

Tables 5.1 and 5.2 present the main classes and methods used in the tool.

Class Name	Description
`Monomial`	$a.\prod_i^n x_i^{k_i}$ where a is a double precision float, k_i is an integer
`Polynomial`	$\sum a.\prod x_i^{k_i}$ a finite sum of `Monomial`(s)
`Polynomial_Constraint`	`Polynomial` p and a constraint type $op \in \{>, <, =\}$
`Polynomial_Region`	a set of `Polynomial_Constraint`(s)
`Continuous_System`	an ordered set of `Polynomial`, if polynomial $p(x)$ is of index i then it is interpreted as $\frac{dx_i}{dt} = p(x)$
`Polynomial_Partition`	an aggregation of a single `Continuous_System` and a set of `Polynomial_region`(s)
`Hybrid_System`	an ordered set of `Continuous_System` (`CS`). Each (`CS`) is assigned transitions containing guards and resets

Table 5.1: Main Classes of the Abstraction Tool

Methods	**Description**
`(obj)_to_string`	where `obj` stands for one of the previous classes name, provides in and out methods to convert a string to object of `obj`, this is implemented by checking the presence of regular expressions using simple deterministic automata.
`polynomial_(operator)`	where `operator` is $(+, *, derive, projection\_flow)$ providing primary operations on polynomials.
`partition(base)`	computes the set of `polynomial_region`(s) using cross product of elements from a set of strings `base` with the number of variables of a `Continuous_System`.
`CAD_Evaluate(string)`	evaluates the $true$ or $false$ nature of a CAD formula using QepCAD.
`Transitions_Compute`	given a `Polynomial_Partition`, this function will compute the formula expressing the scalar product between the dynamics and the normal vector of some boundary while limiting the evaluated boundary to the region of the considered abstract state.

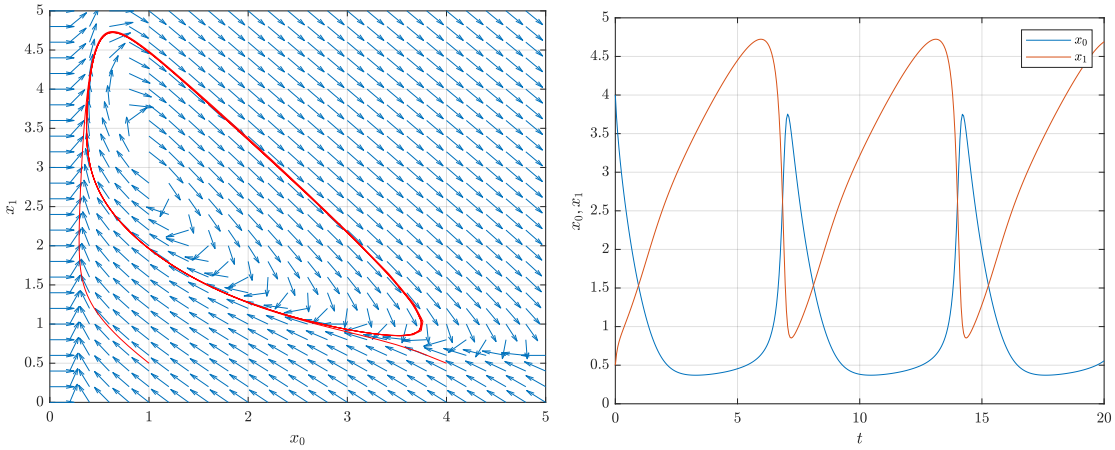
Table 5.2: Main Methods of the Abstraction Tool

5.2 Examples and simulation results

Example 5.1 (Continuous System)**. Cyclic Behavior**
The brusselator is a mathematical model used for representing chemical reactions with cyclic change of color. The dynamics are nonlinear. The model holds a single mode q with a singleton flow F given by the variables derivatives as:

$$\dot{x_0} = 1 - 4x_0 + x_0^2 x_1$$
$$\dot{x_1} = 3x_0 - x_0^2 x_1$$

For illustrative purposes, two numerical simulations of the example show a clockwise rotation of the trajectories with respect to the (x_0, x_1) coordinates (see Figure 5.2). As a first simulation, we are interested in the behavior of the brus-

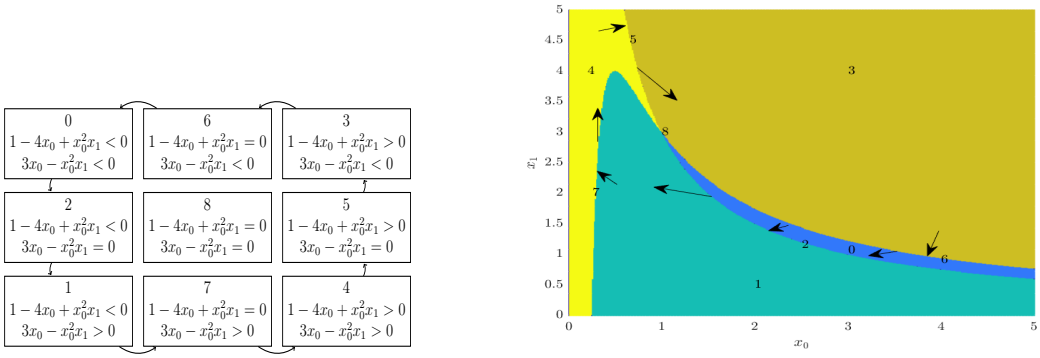


(a) (b)

Figure 5.2: Brusselator normalized phase plane with numerical simulations

selator in the first quadrant of the (x_0, x_1) plane, thus we add to $Inv(q)$ the constraints: $x_0 > 0, x_1 > 0$ as invariants. The system is two-dimensional, hence the computed abstraction according to flow sign vector contains $3^2 = 9$ states. The tool successfully analyzes the example, the produced abstraction is shown both as an automaton view in Figure 5.3.a and in the continuous space view in Figure 5.3.b.

The qualitative simulation shows what behavior of the system is impossible and what behavior is possible. Using a numerical simulator (or computing an



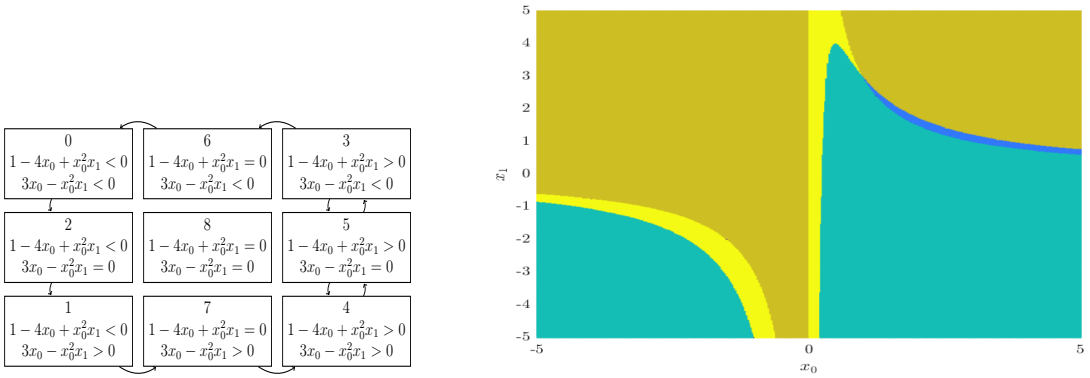
(a) (b)

Figure 5.3: Brusselator computed abstraction with $Inv(q) = (x_0 > 0 \wedge x_1 > 0)$ (a) Automaton view (b) Continuous space view

over-approximating flow-pipe), it would take a number of simulations to show the possible cyclic behavior of the brusselator. Figure 5.3 proves that the rotation direction of the brusselator cycle cannot be counter-clockwise in a single abstraction step and without refinements. This result is valid for all trajectories in the unbounded $(x_0 > 0 \wedge x_1 > 0)$ domain and cannot be proved using numerical or flow-pipe simulations. Note that state 8 corresponds to the equilibrium point, i.e., in a neighborhood around state 8 no trajectories are to reach it in any time. As a result, trajectories come neither in nor out, thus no outgoing or incoming transitions are associated to state 8.

Change of Invariant Another abstraction of the brusselator is computed with a change of the previous invariant from $x_0 > 0 \wedge x_1 > 0$ to *true*, the result is represented in Figure 5.4. Notice that new transitions that were absent appear $(3 \rightarrow 5)$ and $(5 \rightarrow 4)$. The result is correct and is due to the non-connected sets of the considered abstract states. This abstraction shows that the behavior of the system can possibly be counter-clockwise when considering the whole state space $\mathbb{R}^2$.

Refining the abstraction Let us consider more granular regions of the previous partitions for each example. Consider adding $x_1 - 5$ as a refining polynomial of the



(a) (b)

Figure 5.4: Brusselator computed abstraction with $Inv(q) = true$
(a) Automaton view (b) Continuous space view

brusselator initial partition. Figure 5.5 illustrates the newly obtained partition.

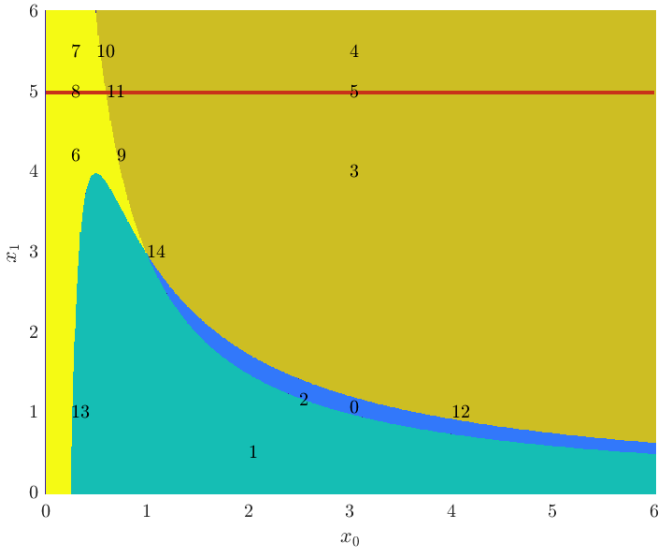


Figure 5.5: Adding a refining polynomial to a previous decomposition

Notice that by applying the refinement algorithm 4.2.4, many of the previously computed transitions remain valid. The regions 3 and 4 and 5 are split respectively as (3, 4, 5) ; (6, 7, 8) and (9, 10, 11). Regions 0, 1, 2, 12, 13, 14 are re-expressed by adding to each of them the containing half-plane of the new refining polynomial. In this case, the half-plane expressed by $x_1 - 5 < 0$ is added to these regions. The

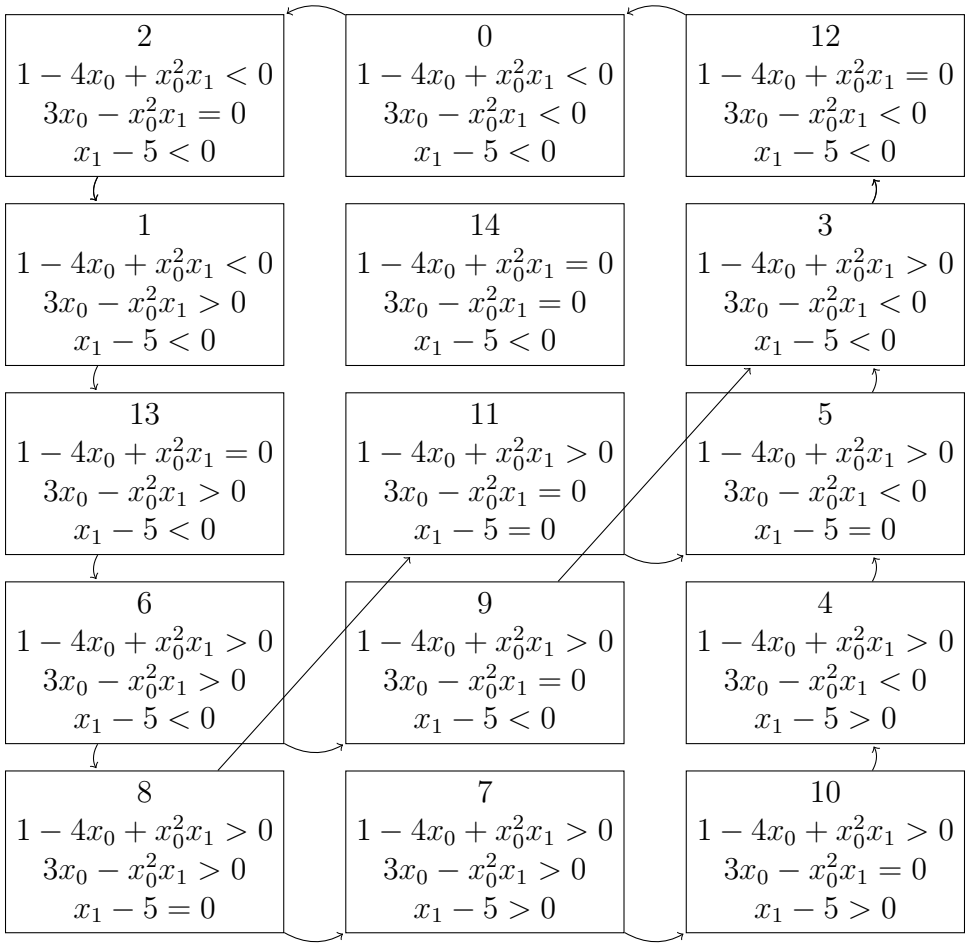


Figure 5.6: Refined Brusselator Abstraction Automaton View

refined abstraction automaton obtained automatically using the tool is observed on Figure 5.6.

5.2.1 Computation Time

Making use of multi-threaded programming It is important to note that the transitions computing can be parallelized, once the abstract states are computed for a single mode. For the moment this parallelization is not implemented in the presented tool. In fact, an evaluation of a set of inequalities using CAD has been proven to be doubly exponential [40]. Hence, parallelizing the computations will significantly reduce the computation time of the abstraction and will allow

one to handle systems with large number of modes and with higher dimensions (i.e., the number of continuous variables involved).

Benchmark evaluation We use the benchmark proposed by Xin Chen in [29] to evaluate computation time over different linear, non-linear and hybrid systems models. The computation time is illustrated in Table 5.3, page 134 and is achieved on a machine equipped with an Intel Core i5-3210M CPU operating at a 2.5 Ghz frequency. All input models are given in appendix A.2 (page 143).

Refined models computation time For some of the previously considered examples from the benchmark, we applied successive refinement steps to obtain a more granular abstraction. The refining polynomials were chosen randomly. The computation time of the refined models and the number of refinement steps (i.e., number of added polynomials for all modes) are illustrated in Table 5.4 (page 135).

Fail reasons Fail$^{(1)}$ occurred with one of the transition computation at the level of the quantifier elimination solver with no specific error. Fail$^{(2)}$ occurred due to the lack of extra cells needed for the computation. In fact the number of cylindrical algebraic decomposition cells in `QepCAD` is capped. From the first transition computation of Biology Model 1, the number of required cells exceeded the maximum cap.

5.2.2 Extending the tool with optimization libraries

Once the qualitative abstraction is computed, the tool can optionally call a non-linear optimization library `Nlopt` to perform optimization operations over the partitioned regions.

Computing V_{min} and V_{max} An important operation that can be performed is computing the maximum and minimum values of the first order derivatives of each continuous variables. After the discrete abstraction of H is computed, the user can choose to compute V_{min} and V_{max}. Similarly to the transitions between regions computation, the computation of the minimal and maximal values of $\dot{x}$ denoted respectively V_{min} and V_{max} can be parallelized as well. The tool interfaces with

the `Nlopt` library and uses existing algorithms to determine safe upper and lower bounds of V_{min} and V_{max} with a tolerated error. The computed values will refine the abstraction and can be represented using rectangular automata.

5.3 Chapter Summary

In this chapter we presented a tool that automatically performs the qualitative simulation of a given non-linear polynomial hybrid automaton given as input. The performed operations can be summarized in two steps: first the computation of the partitions relative to each mode of the hybrid automaton, second the computation of the transitions between each region of the obtained partition. The tool was tested over several examples to evaluate performances.

Model Name	Computation Time (s)	Nbr Variables $\lvert X \rvert$	Nbr Modes $\lvert Q \rvert$	Nbr of Abstract States
Linear Dynamics				
Van Der Pol Oscillator	2.7	2	1	9
Non-Linear Polynomial Dynamics				
Lotka-Volterra	2.9	2	1	9
Buckling Column	2.9	2	1	9
Brusselator	3.2	2	1	9
Lorentz	11.4	3	1	27
Roessler Attractor	8.4	3	1	27
Coupled Van Der Pol	Fail[1]	4	1	
Biology Model 1	Fail[2]	7	1	
Linear Hybrid System				
Bouncing Ball	8	2	1	27
Thermostat	13.3	1	2	10
Non-Linear Polynomial Hybrid System				
2D Example (page. 90)	197	2	2	54
Combined Brusselator	189	2	2	44

Table 5.3: Computation time of the timeless abstraction for different models

Model Name	Computation Time (s)	Nbr Variables $\lvert X\rvert$	Nbr Modes $\lvert Q\rvert$	Nbr of abstract states	Refinement steps
Non-Linear Polynomial Dynamics					
Brusselator	3.2	2	1	9	0
	3.6	2	1	15	1
	10.5	2	1	35	2
	30.1	2	1	54	3
Lorentz	3.6	3	1	27	0
	10.6	3	1	69	1
	30.1	3	1	110	2
	86.4	3	1	152	3
Linear Hybrid System					
Thermostat	4.6	1	2	10	0
	11.1	1	2	12	1
	26.7	1	2	18	2
Non-Linear Polynomial Hybrid System					
Combined Brusselator	76.4	2	2	44	0
	107	2	2	77	1
	346	2	2	117	2

Table 5.4: Applying refinement steps with computation time

Chapter 6

Conclusion

6.1 Results Summary

We were interested in verifying a given formal safety property on a hybrid system, based on discrete abstractions of this system, for which checking this property is decidable and which guarantee that the property is satisfied at the concrete hybrid level if it is satisfied at the abstract level. We focused on the diagnosability property, for its importance in safety analysis at design stage and the challenge it gives rise to. We presented elements from the literature regarding hybrid automata abstractions, however few works handle diagnosability verification, as this property deals with a pair of trajectories and partial observations of the system and is thus more complex to check than reachability. In order to handle time constraints at the abstract level, we chose abstractions of the hybrid automaton as timed automata, related to a decomposition of the state space into geometric regions, the abstract time constraints coming from the estimation of the sojourn time of trajectories in each region. Thus the abstractions over-approximate the regions of interest to which are added time constraints obtained from the dynamics of the concrete system. We adapted a CEGAR scheme for hybrid systems diagnosability verification, based on the counterexample provided at the abstract level by the twin plant based diagnosability checking when diagnosability is proved to be unsatisfied. We presented situations for which the produced counterexample is spurious and a refinement in finer regions and tighter time constraints is then required. A tool prototype that implements the qualitative simulation algorithm

of a given hybrid automaton (with at most polynomial dynamics in terms of expressiveness) have been developed and tested over examples.

6.2 Comparison with the existing literature

6.2.1 Hybrid Automata Abstraction

Works of A. Tiwari [102] The works of Tiwari introduced qualitative reasoning to hybrid automata verification. The broad class of hybrid automata is considered and experimentation is provided through examples and case study. Our work further instantiates the framework to the specific class of polynomial hybrid automata and considers partitions of the state space using semi-algebraic sets. Feasibility and applicability in practice of the qualitative simulation is shown by the tool implementation and a manual refinement of the abstraction is implemented.

Works of C. Sloth [60] Sloth elaborated abstraction methods of hybrid automata represented as timed automata. In his work, inductive invariants are considered and form a basis of the abstraction technique. Consequently, he considers regions of interest for which the projections of the flow to boundaries are always oriented inward, making the region an invariant. This is not the case in our work, where the orientation whether inward or outward is reflected in the abstraction over qualitatively partitioned regions. This is justified as some properties can only be proved by taking all aspects of the flow orientation into account (recall the brusselator example direction of rotation). In our work we propose to combine existing flow-pipe computation techniques with finite state abstractions to obtain a more representative, timed abstraction.

6.2.2 Diagnosability verification

Works of A. Grastien [60] In the abstraction method we previously elaborated, time constraints are used explicitly. When an abstraction refinement is required, tighter time bounds are obtained over the new regions of the refined

decomposition. The proposed abstraction method hence differs from the one proposed by Grastien. In [60], the abstraction consists in retaining some properties of the continuous dynamics, namely mode distinguishability and ephemerality, which are directly checked on the concrete hybrid system when necessary. On the contrary, in our approach the abstractions refer directly to the continuous state space and the continuous dynamics are interpreted with increasing levels of granularity, which results in finer and finer state space decompositions to which time constraints are associated. These abstractions take the form of timed automata.

Works of M. Beneditto [43] Beneditto proposes a timed abstraction to verify diagnosability. The abstraction used is a durational graph, that measures the elapsed time of a trajectory from an initial set until a mode change condition is hit. In our method, the abstraction we use is one that first can be refined, and second is computed qualitatively over partitioned regions. The diagnosability verification is proven via the absence of critical pairs.

6.3 Perspectives

Our thesis work draws many perspectives. The first direct perspective is to elaborate automated rules to implement the refinement strategies by analysis of a diagnosability counterexample given the reasons we previously explained.

Diagnosability specific perspectives

- We wish to extend the diagnosability verification algorithm by attributing an ϵ-precision, for ϵ arbitrary small constant (for a given metrics to be defined), for hybrid automata, guaranteeing the refinement loop termination when the defined precision is achieved. In fact, up to now, we assume a continuous domain for both the values and the time stamps of the observable variables without taking into account sensors precision, i.e., the minimal intervals (of value and of time) that can be captured. This is the reason why our current algorithm may not terminate, due to an infinite refinement process. A fundamental and essential future work perspective is to provide a general algorithm for diagnosability verification with ϵ-precision [69], for ϵ arbitrary

small. This will ensure theoretical termination of the algorithm, as the number of refinement steps to reach the precision will then be finite. And this is actually justified in practice because both model parameters and observations cannot be infinitely accurate, thus the value ϵ for the precision would come from the precision of the model parameters and of the measurements, in space and time.

- In the same spirit, one interesting future work would be to demonstrate a bisimulation relation between the concrete model and the final refined abstract model when considering this minimal precision imposed by the model and the sensors, where the termination of the refinement can thus be guaranteed. In other words, theoretically, we could always deduce the right verdict, either the system is diagnosable or it is not diagnosable with respect to a given minimal precision.

- Another promising aspect of diagnosability verification is to study the potential of this approach for deducing at design stage minimal concrete sets of observations (which events/continuous variables to observe, i.e., which sensors, and at which observation times, e.g., sampling periods of sensors) for which the system is diagnosable (assuming it would be diagnosable if observation was total). This includes both suppressing sensors placement (if not useful otherwise) while keeping existing diagnosability or adding sensors to make diagnosable a non-diagnosable system [22].

- We would like as well to study the application of the approach to the problem of active diagnosis. The latter supposes that the system is controllable via actions at execution time. When some actions are performed the future evolution of the system is changed. If the system is not diagnosable, a controller can be synthesized that forces the system to stay within the diagnosable area of execution. The problem was studied for systems modeled as automata [62].

- Extending the abstraction combined with model checking approach to other formal properties such as predictability seems promising. Predictability is a

property of the model, if satisfied then it is possible to predict in advance the future unavoidable occurrence of a fault, given some sequence of observations.

Tool perspectives

- Interfacing the current tool prototype with existing flow-pipe computation tools to automate the computations of the time bounds of the timed abstraction and for validating/refuting a given counter-example is a crucial step to extend the existing prototype. We plan in particular to use and extend existing tools for timed automata model checking and for over-approximation reachability at the continuous level. We want to investigate also the usage of SMT solvers [31, 30], in particular with theories including ODEs [55], to deal simultaneously with discrete and continuous variables.

- We wish to implement a split of the sign preserving regions, which for the moment can have non-connected components (in the case of polynomial dynamics) to have sign preserving regions with only connected sets. This is feasible in practice, in fact using results from the literature [64] and known theorems such as,

  Theorem 6.1. *Every semialgebraic set has finitely many connected components which are semialgebraic. Every semialgebraic set is locally connected.*

  the obtained abstraction by partitioning the state-space into connected components is guaranteed to stay finite.

- It is also important to interface the tool with a timed abstraction diagnosability model checker which given a fault-modeling partially observable timed automaton, computes the corresponding twin plant and generates (Δ)-critical pairs (such a model checker has been developed at LRI, implemented with the SMT solver Z3). Once implemented it can be interfaced with the current prototype tool to perform CEGAR for verifying automatically diagnosability.

- Implementing a state split strategy guided by a validated counterexample is as well a practical and useful step to perform.

6.4 Publications

International Conferences:

- "Counterexample-Guided Abstraction-Refinement for Hybrid Systems Diagnosability Analysis", H. Zaatiti, L. Ye, P. Dague, J.-P. Gallois, 28th International Workshop on Principles of Diagnosis DX'17, Brescia, Italy, September 2017; long version in the proceedings, Kalpa Publications in Computing, volume 4, pp. 124-143, 2018.
- "Automating Abstraction Computations of Hybrid Systems", H. Zaatiti, L. Ye, P. Dague, J-P. Gallois, Formal Verification of Physical Systems FVPS 2018, workshop of the 11th Conference on Intelligent Computer Mathematics CICM 2018, Hagenberg, Austria, August 2018.

Book chapter:

- "Abstractions Refinement for Hybrid Systems Diagnosability Analysis", H. Zaatiti, L. Ye, P. Dague, J-P Gallois, L. Travé-Massuyès, in Diagnosability, Security and Safety of Hybrid Dynamic and Cyber-Physical Systems, Springer, 2018.

Appendix A

Tool Grammar and Models

A.1 Tool Grammar

The input model must satisfy the following input grammar rules. The grammar is straightforward allowing to textually describe a hybrid automaton with at most polynomial dynamics:

```
model_name
{
     INIT
    {
      (mode_number, poly)
    }
    [mode_number
    {
       (ODE;)*
       INVARIANT:
       (inv;)*
       REFINE:
       (refine;)*
       TRANS:
      →mode_number, G: (guard;)*, R: (reset;)*;
    }]*
}
```

Figure A.1: Model Input Grammar

$(xyz)^*$ or $[xyz]^*$ refers to any number of repetitions of xyz. *mode_number* is an integer from 0 up to the number of modes of the hybrid automaton.

A.2 Input models

A.2.1 Van Der Pol Oscillator

$$\dot{x_0} = x_1$$
$$\dot{x_1} = x_1 - x_0 - x_0^2 x_1$$

A.2.2 Lotka Volterra

$$\dot{x_0} = 2x_0 - x_0 x_1$$
$$\dot{x_1} = -3x_1 + x_0 x_1$$

A.2.3 Buckling Column

$$\dot{x_0} = -x_1$$
$$\dot{x_1} = 2x_0 - x_0^3 - x_0 + 1$$

A.2.4 Lorentz

$$\dot{x_0} = -10x_0 + 10x_1$$
$$\dot{x_1} = 28x_0 - x_1 - x_0 x_1$$
$$\dot{x_2} = -3x_2 + x_2 x_1$$

A.2.5 Roesller Attractor

$$\dot{x_0} = -x_1 - x_2$$
$$\dot{x_1} = x_0 + x_1$$
$$\dot{x_2} = 1 + x_2 x_0 - 6x_2$$

A.2.6 Coupled Van Der Pol Oscillator

$$\begin{aligned}
\dot{x_0} &= x_1 \\
\dot{x_1} &= x_1 - x_0^2 x_1 + x_2 - 2x_0 \\
\dot{x_2} &= x_3 \\
\dot{x_3} &= x_3 - x_2^2 x_3 + x_0 - 2x_2
\end{aligned}$$

A.2.7 Biology Model 1

$$\begin{aligned}
\dot{x_0} &= -x_0 + 5x_2x_3 \\
\dot{x_1} &= x_0 - x_1 \\
\dot{x_2} &= x_1 - 5x_2x_3 \\
\dot{x_3} &= 5x_4x_5 - 5x_2x_3 \\
\dot{x_4} &= -5x_4x_5 + 5x_2x_3 \\
\dot{x_5} &= x_6 - 5x_4x_5 \\
\dot{x_6} &= -x_6 + 5x_4x_5
\end{aligned}$$

A.2.8 Bouncing Ball

Mode q_0:

$$\begin{aligned}
\dot{x_0} &= x_1 \\
\dot{x_1} &= -10 \\
Inv(q_0) &= \{x_0 > 0\}
\end{aligned}$$

Transitions: From q_0 to q_0 guarded by $G : x_0 < 0$.

A.2.9 Combined Brusselator

Mode q_0:

$$\dot{x_0} = 1 - 4x_0 + x_0^2 x_1$$
$$\dot{x_1} = 3x_0 - x_0^2 x_1$$
$$Inv(q_0) = \{x_0 > 0, x_1 > 0\}$$

Transitions: From q_0 to q_1 guarded by $G : x_1 < x_0$.

Mode q_1:

$$\dot{x_0} = 1 - 2x_0 + 2x_0^2 x_1$$
$$\dot{x_1} = x_0 - 2x_0^2 x_1$$
$$Inv(q_0) = \{x_0 > 0, x_1 > 0\}$$

Transitions: From q_1 to q_0 guarded by $G : x_0 < x_1$.

Appendix B

Discussion on critical points

Finite number of critical points The derivatives along each coordinate are all null at the same time in a finite number of points X_{eq} of region p, we repartition p such that each new region p_i has exactly one point of X_{eq} on its boundary. In one dimension, suppose the null derivative point is at x_{eq} and the initial value of $x(t)$ is x_0. The time, which we denote by $\mathcal{T}$, taken by the continuous variable $x(t)$ to cross from x_0 to x_{eq} can in some cases be finite. If $\dot{x}$ is of the form x^k where $k \in \mathbb{R}$, then by studying $\mathcal{T}$ in a neighborhood around $k = 1$ we obtain:

$$\mathcal{T} = \int_0^{t_{eq}} \mathrm{d}t = \int_{x_0}^{x_{eq}} \frac{1}{\dot{x}} \mathrm{d}x = \int_{x_0}^{0} \frac{1}{x^k} \mathrm{d}x = \left[\frac{x^{-k+1}}{(-k+1)} \right]_{x_0}^{0} \tag{B.1}$$

$\mathcal{T}$ is convergent if $k < 1$. Thus a time bound can be computed for all dynamics of the form $\dot{x} = x^k$ with $k < 1$, for example for the square root $\dot{x} = \sqrt{x}$. In two dimensions, let r be the distance from the equilibrium point $M_{eq}(x_{eq}, y_{eq})$ to a point $M(x, y)$ which is initially in region p. Let $X = x_{eq} - x$ and $Y = y_{eq} - y$. Since $r^2 = X^2 + Y^2$ then $2r\dot{r} = 2X\dot{X} + 2Y\dot{Y}$ and if $\dot{r} \neq 0$ the time to reach M_{eq} is :

$$\mathcal{T} = \int_0^{t_{eq}} \mathrm{d}t = \int_{r_0}^{0} \frac{1}{\dot{r}} \mathrm{d}r = \int_{r_0}^{0} \frac{r}{X\dot{X} + Y\dot{Y}} \mathrm{d}r \tag{B.2}$$

Example B.1. Consider the two dimensional continuous system $\dot{x} = -x^2$ and $\dot{y} = -y$ where $(x, y) \in \mathbb{R}^+ \times \mathbb{R}^+$. The equilibrium point is $M_{eq}(0, 0)$. In polar coordinates $x = rcos(\theta)$ and $y = rsin(\theta)$, then $r\dot{r} = x\dot{x} + y\dot{y} = -x^3 - y^2 = -x^2 - y^2 + x^2 - x^3 = -r^2 + x^2(1 - x) = r^2(-1 + cos^2(\theta)(1 - rcos(\theta)))$. In a

neighborhood around $(0, 0)$:

$$\mathcal{T} = \int_{r_0}^{0} \frac{1}{r(-1 + cos^2(\theta)(1 - rcos(\theta)))} \, \mathrm{d}r \geq \int_{r_0}^{0} \frac{-1}{r} \, \mathrm{d}r \tag{B.3}$$

Thus $\mathcal{T}$ is infinite, the equilibrium point is never reached. This reasoning can be extended to dimension n by evaluating $r\dot{r}$ and using spherical (or hyperspherical) coordinates and to polynomial with real exponents. We can take an example of square root, for instance $\dot{x} = \sqrt{x}$ and demonstrate the time $\mathcal{T}$ is finite, then the equilibrium is reached.

Infinite number of null derivatives. Studying a case where at least one derivative along an axis takes an infinite number of null values in a connected set can be done by extending the previous method. For the particular class of continuous systems where the dynamics are only allowed multi-affine function form, [80] showed how it is possible to capture time constraints by decomposing the infinite state space $\mathbb{R}^n$ into hypercubes and evaluate the time elapsed between entering and exiting each cube by bounding the dynamics.

Brusselator time bounds. For the Example 5.1 (page 127) of the brusselator dynamics, which is a repeller case around the point $M_0 = (1, 3)$, consider the ring set R that excludes M_0 such that $R = \{(x_0, x_1) \mid 0.09 < (-1 + x_0)^2 + (-3 + x_1)^2 < 0.5625\}$. Let $v = \sqrt{\dot{x_0}^2 + \dot{x_1}^2}$ then $v^2 = (1-4x_0+x_0^2x_1)^2+(3x_0-x_0^2x_1)^2$ and, using a solver, we compute a lower bound $v_{low}^2 = 0.0051$ of v^2. It has been proven that all trajectories initially in $S_0 = \mathbb{R}^2 \setminus \{M_0\}$ converge towards a fixed orbit of the phase plane contained within R. Suppose we split the region R into two connected sets R_1 and R_2 such that $R = R_1 \biguplus R_2$, for each of which the maximal sojourn duration t_{max} is a positive real constant. This is possible since v admits a lower bound v_{low}. This states that all trajectories initiated from S_0 will cross R_1 and R_2 sequentially infinitely often. In practice, the presence of the system in either R_1 or R_2 can correspond to two different visible colors of the chemical reaction.

Bibliography

[1] S. B. Akers. Binary decision diagrams. *IEEE Transactions on computers*, (6):509–516, 1978.

[2] M. Althoff, D. Grebenyuk, and N. Kochdumper. Implementation of taylor models in cora 2018. In *Proc. of the 5th International Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 145–173, 2018.

[3] M. Althoff, O. Stursberg, and M. Buss. Computing reachable sets of hybrid systems using a combination of zonotopes and polytopes. *Nonlinear analysis: hybrid systems*, 4(2):233–249, 2010.

[4] R. Alur, T. Dang, and F. Ivančić. Reachability analysis of hybrid systems via predicate abstraction. In C. J. Tomlin and M. R. Greenstreet, editors, *Hybrid Systems: Computation and Control*, pages 35–48, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[5] R. Alur, T. Dang, and F. Ivančić. Progress on reachability analysis of hybrid systems using predicate abstraction. In *International Workshop on Hybrid Systems: Computation and Control*, pages 4–19. Springer, 2003.

[6] R. Alur, T. Dang, and F. Ivančić. Counterexample-guided predicate abstraction of hybrid systems. *Theor. Comput. Sci.*, 354(2):250–271, Mar. 2006.

[7] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[8] G. A.Tiwari. Series of abstractions for hybrid automata. *Hybrid Systems: Computation and Control*, 2289:465–478, 2002.

[9] S. Bak. Reducing the wrapping effect in flowpipe construction using pseudo-invariants. In *Proceedings of the 4th ACM SIGBED International Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems*, pages 40–43. ACM, 2014.

[10] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanovi'c, T. King, A. Reynolds, and C. Tinelli. CVC4. In G. Gopalakrishnan and S. Qadeer, editors, *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV '11)*, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, July 2011. Snowbird, Utah.

[11] M. Basseville, M. Kinnaert, and M. Nyberg. On fault detectability and isolability. *European Journal of Control*, 7(6):625–641, 2001.

[12] S. Basu, R. Pollack, and M.-F. Roy. Complexity of computing semi-algebraic descriptions of the connected components of a semi-algebraic set. In *Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation*, ISSAC '98, pages 25–29, New York, NY, USA, 1998. ACM.

[13] M. Bayoudh, L. Travé-Massuyès, and X. Olive. Hybrid systems diagnosability by abstracting faulty continuous dynamics. In *Proceedings of the 17th International Principles of Diagnosis Workshop*, pages 9–15. Citeseer, 2006.

[14] M. Bayoudh, L. Travé-Massuyès, and X. Olive. Coupling continuous and discrete event system techniques for hybrid systems diagnosability analysis. In *Proceedings of the 18th European Conference on Artificial Intelligence ECAI*, pages 219–223, Patras (Greece), 2008.

[15] M. Bayoudh, L. Travé-Massuyès, and X. Olive. Active diagnosis of hybrid systems guided by diagnosability properties. *IFAC Proceedings Volumes*, 42(8):1498–1503, 2009.

[16] M. Bayoudh and L. Travé-Massuyès. Diagnosability analysis of hybrid systems cast in a discrete-event framework. *Discrete Event Dynamics Systems*, 24(3):309–338, 2014.

[17] L. Benvenuti, D. Bresolin, A. Casagrande, P. Collins, A. Ferrari, E. Mazzi, A. Sangiovanni-Vincentelli, and T. Villa. Reachability computation for hybrid systems with ariadne. *{IFAC} Proceedings Volumes*, 41(2):8960 – 8965, 2008. 17th {IFAC} World Congress.

[18] S. Biswas, D. Sarkar, S. Mukhopadhyay, and A. Patra. Diagnosability analysis of real time hybrid systems. In *Proceedings of the IEEE International Conference on Industrial Technology ICIT'06*, pages 104–109, Mumbai, India, 2006.

[19] O. Botchkarev and S. Tripakis. Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations. In *Proceedings of the 3rd International Workshop on Hybrid Systems: Computation and Control (HSCC'00)*, volume 1790 of *LNCS*, pages 73–88. Springer, 2000.

[20] A. Bouajjani, R. Echahed, and J. Sifakis. On model checking for real-time properties with durations. In *Logic in Computer Science, 1993. LICS'93., Proceedings of Eighth Annual IEEE Symposium on*, pages 147–159. IEEE, 1993.

[21] P. Bouyer, F. Chevalier, and D. D'Souza. Fault diagnosis using timed automata. In *International Conference on Foundations of Software Science and Computation Structures*, pages 219–233. Springer, 2005.

[22] L. Brandán Briones, A. Lazovik, and P. Dague. Optimizing the system observability level for diagnosability. In *Proceedings of the 3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA'08)*, Chalkidiki, Kassandra, Greece, 2008.

[23] J. Chen and R. Patton. A re-examination of the relationship between. parity space and observer- based approaches in fault diagnosis. In *In Proceedings of the IFAC Symposium on Fault Detection, Supervision and Safety of Technical Systems Safeprocess'94*, pages 590–596, Helsinki, Finland, 1994.

[24] X. Chen. *Reachability Analysis of Non-Linear Hybrid Systems Using Taylor Models*. PhD thesis, Aachen University, 2015.

[25] X. Chen, E. Abraham, and S. Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd*, pages 183–192. IEEE, 2012.

[26] X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification*, pages 258–263. Springer, 2013.

[27] X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification*, pages 258–263. Springer, 2013.

[28] X. Chen, S. Mover, and S. Sankaranarayanan. Compositional relational abstraction for nonlinear hybrid systems. *ACM Trans. Embed. Comput. Syst.*, 16(5s):187:1–187:19, Sept. 2017.

[29] X. Chen, S. Schupp, I. B. Makhlouf, E. Ábrahám, G. Frehse, and S. Kowalewski. A benchmark suite for hybrid systems reachability analysis. In *NASA Formal Methods Symposium*, pages 408–414. Springer, 2015.

[30] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta. HyComp: An SMT-based model checker for hybrid systems. In *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS-2015)*, pages 52–67, London, UK, 2015.

[31] A. Cimatti, S. Mover, and S. Tonetta. SMT-based scenario verification for hybrid systems. *Formal methods in system design*, 42(1):46–66, 2013.

[32] A. Cimatti, C. Pecheur, and R. Cavada. Formal Verification of Diagnosability via Symbolic Model Checking. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 363–369. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence, Inc., 2003.

[33] E. Clarke, A. Biere, R. Raimi, and Y. Zhu. Bounded model checking using satisfiability solving. *Form. Methods Syst. Des.*, 19(1):7–34, July 2001.

[34] E. M. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT press, 1999.

[35] V. Cocquempot, T. E. Mezyani, and M. Staroswiecki. Fault detection and isolation for hybrid systems using structured parity residuals. In *Proceedings of the IEEE/IFAC-ASCC: Asian Control Conference*, volume 2, pages 1204–1212, Melbourne, Australia, 2004.

[36] M. Daigle, X. Koutsoukos, and G. Biswas. An event-based approach to hybrid systems diagnosability. In *Proceedings of the 19th International Workshop on Principles of Diagnosis*, pages 47–54. Citeseer, 2008.

[37] M. J. Daigle. *A qualitative event-based approach to fault diagnosis of hybrid systems*. PhD thesis, Vanderbilt University, 2008.

[38] M. J. Daigle, D. Koutsoukos, and G. Biswas. An event-based approach to integrated parametric and discrete fault diagnosis in hybrid systems. *Transactions of the Institute of Measurement and Control, Special Issue on Hybrid and Switched Systems*, 32(5):487–510, 2010.

[39] M. J. Daigle, I. Roychoudhury, G. Biswas, D. Koutsoukos, A. Patterson-Hine, and S. Poll. A comprehensive diagnosis methodology for complex hybrid systems: A case study on spacecraft power distribution systems. *IEEE Transactions of Systems, Man, and Cybernetics, Part A, Special Issue on Model-based Diagnosis: Facing Challenges in Real-world Applications*, 4(5):917–931, 2010.

[40] J. H. Davenport and J. Heintz. Real quantifier elimination is doubly exponential. *Journal of Symbolic Computation*, 5(1):29 – 35, 1988.

[41] L. De Moura and N. Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[42] Y. Deng, A. D'Innocenzo, M. D. Di Benedetto, S. Di Gennaro, and A. A. Julius. Verification of hybrid automata diagnosability with measurement uncertainty. *IEEE Transactions on Automatic Control*, 61(4):982–993, 2016.

[43] M. D. Di Benedetto, S. Di Gennaro, and A. D'Innocenzo. Verification of hybrid automata diagnosability by abstraction. *IEEE Transactions on Automatic Control*, 56(9):2050–2061, 2011.

[44] O. Diene, M. V. Moreira, V. R. Alvarez, and E. R. Silva. Computational methods for diagnosability verification of hybrid systems. In *Control Applications (CCA), 2015 IEEE Conference on*, pages 382–387. IEEE, 2015.

[45] O. Diene, E. R. Silva, and M. V. Moreira. Analysis and verification of the diagnosability of hybrid systems. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 1–6. IEEE, 2014.

[46] S. Ding. *Model-based fault diagnosis techniques: design schemes, algorithms, and tools*. Springer Science & Business Media, 2008.

[47] A. Djaballah, A. Chapoutot, M. Kieffer, and O. Bouissou. Construction of parametric barrier functions for dynamical systems using interval analysis. *Automatica*, 78:287–296, 2017.

[48] C. Edmund, F. Ansgar, H. Zhi, K. Bruce, S. Olaf, and T. Michael. Verification of hybrid systems based on counterexample-guided abstraction refinement. In H. Garavel and J. Hatcliff, editors, *Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS-2003)*, volume 2619 of *LNCS*, pages 192–207. Springer, 2003.

[49] A. Eggers, N. Ramdani, N. Nedialkov, and M. Fränzle. Improving sat modulo ode for hybrid systems analysis by combining different enclosure methods. In *Proceedings of the 9th International Conference on Software Engineering and Formal Methods*, SEFM'11, pages 172–187, Berlin, Heidelberg, 2011. Springer-Verlag.

[50] B. Falkenhainer and K. D. Forbus. Cmpositional modeling: finding the right model for the job. *Artificial intelligence*, 51(1-3):95–143, 1991.

[51] G. Fourlas, K. Kyriakopoulos, and N. Krikelis. Diagnosability of hybrid systems. In *Proceedings of the 10th Mediterranean Conference on Control and Automation-MED2002*, pages 3994–3999, Lisbon, Portugal, 2002.

[52] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. Spaceex: Scalable verification of hybrid systems. In S. Q. Ganesh Gopalakrishnan, editor, *Proc. 23rd International Conference on Computer Aided Verification (CAV)*, LNCS. Springer, 2011.

[53] J.-P. Gallois and J.-Y. Pierron. Qualitative simulation and validation of complex hybrid systems. In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, TOULOUSE, France, Jan. 2016.

[54] S. Gao, S. Kong, and E. Clarke. Satisfiability modulo odes. *arXiv preprint arXiv:1310.8278*, 2013.

[55] S. Gao, S. Kong, and E. Clarke. Satisfiability modulo ODEs. *Formal Methods in Computer-Aided Design (FMCAD)*, 2013.

[56] E. B. C. G.E Collins and J.R.Johnson. Quantifier elimination by cylindrical algebraic decomposition. pages 8–23, 1998.

[57] V. Germanos, S. Haar, V. Khomenko, and S. Schwoon. Diagnosability under weak fairness. In *Proceedings of the 14th International Conference on Application of Concurrency to System Design (ACSD'14)*, Tunis, Tunisia, june 2014. IEEE Computer Society Press.

[58] J. Gertler. *Fault Detection and Diagnosis in Engineering Systems*. Marcel Deker, 1998.

[59] A. Grastien. Symbolic testing of diagnosability. In *20th International Workshop on Principles of Diagnosis (DX-09)*, pages 131–138, 2009.

[60] A. Grastien, L. Travé-Massuyès, and V. c Puig. Solving diagnosability of hybrid systems via abstraction and discrete event techniques. In *Proceedings of the 27th International Principles of Diagnosis Workshop*, 2016.

[61] S. Gulwani and A. Tiwari. Constraint-based approach for analysis of hybrid systems. In *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA, July 7-14, 2008, Proceedings*, pages 190–203, 2008.

[62] S. Haar, S. Haddad, T. Melliti, and S. Schwoon. Optimal constructions for active diagnosis. *Journal of Computer and System Sciences*, 83(1):101–120, 2017.

[63] E. Hainry. Decidability and undecidability in dynamical systems. In *Research Report*, 2009.

[64] J. Heintz, M.-F. Roy, and P. Solernó. Description of the connected components of a semialgebraic set in single exponential time. *Discrete & Computational Geometry*, 11(2):121–140, Feb 1994.

[65] T. A. Henzinger. The theory of hybrid automata. In *Verification of Digital and Hybrid Systems*, pages 265–292. Springer, 2000.

[66] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *Journal of Computer and System Sciences*, pages 373–382. ACM Press, 1995.

[67] S. Jiang, Z. Huang, V. Chandra, and R. Kumar. A polynomial algorithm for testing diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 46(8):1318–1321, 2001.

[68] E. Kilic. Diagnosability of fuzzy discrete event systems. *Information Sciences*, 178(3):858–870, 2008.

[69] K.-D. Kim, S. Mitra, and P. R. Kumar. Computing bounded epsilon-reach set with finite precision computations for a class of linear hybrid automata. In *Proceedings of the ACM International Conference on Hybrid Systems: Computation and Control*, 2011.

[70] M. Kloetzer and C. Belta. Reachability analysis of multi-affine systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 348–362. Springer, 2006.

[71] H. Kong, F. He, X. Song, W. N. Hung, and M. Gu. Exponential-condition-based barrier certificate generation for safety verification of hybrid systems. In *International Conference on Computer Aided Verification*, pages 242–257. Springer, 2013.

[72] S. Kong, S. Gao, W. Chen, and E. Clarke. dreach: δ-reachability analysis for hybrid systems. In *International Conference on TOOLS and Algorithms for the Construction and Analysis of Systems*, pages 200–205. Springer, 2015.

[73] B. Kuipers. Qualitative simulation. In *Readings in qualitative reasoning about physical systems*, pages 236–260. Elsevier, 1990.

[74] B. Kuipers. *Qualitative reasoning: modeling and simulation with incomplete knowledge*. MIT Press, Cambridge, Massachusetts, USA, 1994.

[75] R. Lal and P. Prabhakar. Bounded error flowpipe computation of parameterized linear systems. In *Proceedings of the 12th International Conference on Embedded Software*, pages 237–246. IEEE Press, 2015.

[76] E. A. Lee. The past, present and future of cyber-physical systems: A focus on models. *Sensors*, 15(3):4837–4869, 2015.

[77] F. Liu and D. Qiu. Safe diagnosability of stochastic discrete-event systems. *IEEE Transactions on Automatic Control*, 53(5):1291–1296, 2008.

[78] P. D. Louise Travé-Massuyès. *Modèles et raisonnements qualitatifs*. Hermes, 10 2003.

[79] O. Maler. Algorithmic verification of continuous and hybrid systems. *arXiv preprint arXiv:1403.0952*, 2014.

[80] O. Maler and G. Batt. Approximating continuous systems by timed automata. In *Formal methods in systems biology*, pages 77–89. Springer, 2008.

[81] A. S. Matveev and A. V. Savkin. *Qualitative Theory of Hybrid Dynamical Systems*. Birkhauser Boston, 2000.

[82] K. L. McMillan. Symbolic model checking. In *Symbolic Model Checking*, pages 25–60. Springer, 1993.

[83] T. Melliti and P. Dague. Generalizing diagnosability definition and checking for open systems: a Game structure approach. In *Proceedings of the 21st International Workshop on Principles of Diagnosis DX'10*, pages 103–110, Portland (OR), United States, 2010.

[84] I. M. Mitchell, A. M. Bayen, and C. J. Tomlin. A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games. *IEEE Transactions on automatic control*, 50(7):947–957, 2005.

[85] S. Mover, A. Cimatti, A. Tiwari, and S. Tonetta. Time-aware relational abstractions for hybrid systems. In *Proceedings of the Eleventh ACM International Conference on Embedded Software*, EMSOFT '13, pages 14:1–14:10, Piscataway, NJ, USA, 2013. IEEE Press.

[86] T. Nishida. Grammatical description of behaviors of ordinary differential equations in two-dimensional phase space. *Artificial Intelligence*, 91(1):3 – 32, 1997.

[87] M. Nyberg. Criterions for detectability and strong detectability of faults in linear systems. *International Journal of Control*, 75(7):490–501, 2002.

[88] U. of Verona, U. of Maastricht, U. of Udine, and A. laboratory. Ariadne: An open tool for hybrid system analysis. http://trac.parades.rm.cnr.it/ariadne/, 2016. [Online; accessed 06-October-2016].

[89] Y. Pencolé. Diagnosability Analysis of Distributed Discrete Event Systems. In *Proceedings of the 16th European Conference on Articifial Intelligent (ECAI04)*, pages 43–47. Nieuwe Hemweg: IOS Press., 2004.

[90] Y. Pencolé and A. Subias. A chronicle-based diagnosability approach for discrete timed-event systems: Application to web-services. *Journal of Universal Computer Science*, 15(17):3246–3272, 2009.

[91] A. Platzer. Differential dynamic logic for verifying parametric hybrid systems. In N. Olivetti, editor, *TABLEAUX*, volume 4548 of *LNCS*, pages 216–232. Springer, 2007.

[92] A. Platzer. Keymaera. http://symbolaris.com/info/KeYmaera.html, 2016. [Online; accessed 28-September-2016].

[93] P. Ribot and Y. Pencolé. Design requirements for the diagnosability of distributed discrete event systems. In *Proc. 19th Intl. Workshop on Principles of Diagnosis (DX), Blue Mountains, Australia*, pages 347–354, 2008.

[94] J. Rintanen. Diagnosers and diagnosability of succinct transition systems. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 538–544, Hyderabad, India, 2007.

[95] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of Discrete Event System. *Transactions on Automatic Control*, 40(9):1555–1575, 1995.

[96] S. Sankaranarayanan and A. Tiwari. Relational abstractions for continuous and hybrid systems. In *Proceedings of the 23rd International Conference on Computer Aided Verification*, CAV'11, pages 686–702, Berlin, Heidelberg, 2011. Springer-Verlag.

[97] A. Schumann and J. Huang. A Scalable Jointree Algorithm for Diagnosability. In *Proceedings of the 23rd American National Conference on Artificial Intelligence (AAAI-08)*, pages 535–540. Menlo Park, Calif.: AAAI Press., 2008.

[98] C. Sloth, G. J. Pappas, and R. Wisniewski. Compositional safety analysis using barrier certificates. In *Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control*, pages 15–24. ACM, 2012.

[99] C. Sloth and R. Wisniewski. Complete abstractions of dynamical systems by timed automata. *Nonlinear Analysis: Hybrid Systems*, 7(1):80–100, 2013.

[100] W. Taha, A. Duracz, Y. Zeng, A. Kevin, P. Brauner, J. Duracz, F. Xu, R. Cartwright, M. Konecny, J. Inoue, A. Sant'Anna, R. Philippsen, A. Chapoutot, M. O'Malley, A. Ames, V. Gaspes, L. Hvatum, S. Mehta, H. Eriksson, and C. Grante. Acumen: An Open-Source Testbed for Cyber-Physical Systems Research. In B. Mandler, J. Marquez-Barja, R.-L. Vieriu, M. E. M. Campista, D. Cagáňová, H. Chaouchi, S. Zeadally, M. Badra, S. Giordano, M. Fazio, and A. Somov, editors, *International Internet of Things Summit*, volume 169, pages 118–130, Rome, Italy, Oct. 2015. Springer.

[101] D. Thorsley and D. Teneketzis. Diagnosability of stochastic discrete-event systems. *IEEE Transactions on Automatic Control*, 50(4):476–492, 2005.

[102] A. Tiwari. Abstractions for hybrid systems. *Formal Methods in Systems Design*, 32:57–83, 2008.

[103] L. Travé-Massuyès, M. Cordier, and X. Pucel. Comparing diagnosability criterions in continuous systems and descrete events systems. In *Proceedings of the 6th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes Safeprocess'06*, pages 55–60, Beijing, P.R. China, 2006.

[104] L. Travé-Massuyès, T. Escobet, and X. Olive. Diagnosability analysis based on component-supported analytical redundancy relations. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 36(6):1146–1160, 2006.

[105] S. Tripakis. Fault diagnosis for timed automata. In *Proceedings of International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT-2002)*, volume 2469 of *LNCS*, pages 205–221. Springer, 2002.

[106] Y. Yan, L. Ye, and P. Dague. Diagnosability for Patterns in Distributed Discrete Event Systems. In *21st International Workshop on Principles of Diagnosis DX'10*, pages 345–352, Portland, OR États-Unis, 2010.

[107] L. Ye and P. Dague. Diagnosability Analysis of Discrete Event Systems with Autonomous Components. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI-10)*, pages 105–110. Nieuwe Hemweg: IOS Press., 2010.

[108] T.-S. Yoo and S. Lafortune. Polynomial-time verification of diagnosability of partially observed discrete-event systems. *IEEE Transactions on Automatic Control (TAC)*, 47(9):1491–1495, 2002.

[109] H. Zaatiti, L. Ye, P. Dague, and J.-P. Gallois. Automating Abstraction Computations of Hybrid Systems. In *CICM 2018 - 11th Conference on Intelligent Computer Mathematics ; Workshop FVPS 2018 - Formal Verification of Physical Systems*, Hagenberg, Austria, Aug. 2018.

[110] H. Zaatiti, L. Ye, P. Dague, and J.-P. Gallois. Counterexample-guided abstraction-refinement for hybrid systems diagnosability analysis. In M. Zanella, I. Pill, and A. Cimatti, editors, *28th International Workshop on Principles of Diagnosis (DX'17)*, volume 4 of *Kalpa Publications in Computing*, pages 124–143. EasyChair, 2018.

[111] H. Zaatiti, L. Ye, P. Dague, J.-P. Gallois, and L. Travé-Massuyès. *Abstractions Refinement for Hybrid Systems Diagnosability Analysis*, pages 279–318. Springer International Publishing, Cham, 2018.

[112] J. Zaytoon and S. Lafortune. Overview of fault diagnosis methods for discrete event systems. *Annual Reviews in Control*, 37(2):308–320, 2013.

[113] J. Zhang, K. H. Johansson, J. Lygeros, and S. Sastry. Zeno hybrid systems. *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, 11(5):435–451, 2001.

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre: Modélisation et Simulation Qualitatives des Systèmes Hybrides

Mots clés: Automates Hybrides, Abstraction, Raisonnement Qualitative, Diagnosticabilité.

Résumé: Les systèmes hybrides sont des systèmes complexes qui combinent un double comportement continu et discret. La vérification des propriétés comportementale comme la sûreté de ces systèmes, depuis la modélisation ou durant leur exécution en ligne, est une tâche difficile. En effet, calculer l'ensemble des états atteignables par un système hybride est indécidable. Une façon de vérifier ces propriétés est de calculer des abstractions discrètes et d'inférer le résultat obtenu du système abstrait vers le système d'origine. Dans ce travail on est concerné par des abstractions orientées pour vérifier la diagnosticabilité d'un système hybride donné. Notre objectif est de créer des abstractions discrètes pour vérifier si une faute qui peut survenir durant l'exécution du système peut être détecter sans ambiguïté en un temps borné par le diagnostiqueur. La vérification est effectuée sur l'abstraction en utilisant des méthodes classiques, développées pour les systèmes à évènements discrets qui fournissent un contre-exemple si le système n'est pas diagnosticable. L'absence d'un tel contre-exemple prouve la diagnosticabilité du système original. En présence d'un contre-exemple, la première étape consiste à vérifier si le contre-exemple est un artefact résultant de l'abstraction et existe au niveau du système hybride, témoignant ainsi de la non-diagnosticabilité du système. Dans le cas-contraire, on montre comment raffiner l'abstraction et continuer à rechercher un autre contre-exemple.

Title: Modeling and Qualitative Simulation of Hybrid Systems

Keywords: Hybrid Automata, Abstraction, Qualitative Reasoning, Diagnosability.

Abstract: Hybrid systems are complex systems that combine both discrete and continuous behaviors. Verifying behavioral or safety properties of such systems, either at design stage or on-line is a challenging task. Actually, computing the reachable set of states of a hybrid system is undecidable. One way to verify those properties over such systems is by computing discrete abstractions and inferring them from the abstract system back to the original system. We are concerned with abstractions oriented towards hybrid systems diagnosability checking. Our goal is to create discrete abstractions in order to verify if a fault that would occur at runtime could be unambiguously detected in finite time by the diagnoser. This verification can be done on the abstraction by classical methods developed for discrete event systems, which provide a counterexample in case of non-diagnosability. The absence of such a counterexample proves the diagnosability of the original hybrid system. In the presence of a counterexample, the first step is to check if it is not a spurious effect of the abstraction and actually exists for the hybrid system, witnessing thus non-diagnosability. Otherwise, we show how to refine the abstraction and continue the process of looking for another counterexample.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France