THÈSE DE DOCTORAT

Présentée à l'École Normale Supérieure de Cachan

par Romain SOULAT

en vue de l'obtention du grade de

Docteur de l'École Normale Supérieure de Cachan

Spécialité informatique

Synthesis of Correct-by-Design Schedulers for Hybrid Systems

Soutenue le 18/02/2014 à Cachan devant un jury composé de :

Étienne	ANDRÉ	Examinateur
Franck	CASSEZ	Rapporteur
Laurent	FRIBOURG	Directeur de thèse
Antoine	GIRARD	Examinateur
Éric	GOUBAULT	Rapporteur
Luc	JAULIN	Examinateur

Laboratoire Spécification et Vérification
École Normale Supérieure de Cachan/CNRS/UMR 8643
61 avenue du Président WILSON, 94235 Cachan Cedex, France

Abstract

In this thesis, we are interested in designing schedulers for hybrid systems. We consider two specific subclasses of hybrid systems, real-time systems where tasks are competing for the access to common resources, and sampled switched systems where a choice has to be made on dynamics of the system to reach goals.

Scheduling consists in defining the order in which the tasks will be run on the processors in order to complete all the tasks before a given deadline. In the first part of this thesis, we are interested in the scheduling of periodic tasks on multiprocessor architectures. We are especially interested in the robustness of schedulers, i.e., to prove that some values of the system parameters can be modified, and until what value they can be extended while preserving the scheduling order and meeting the deadlines. The Inverse Method can be used to prove the robustness of parametric timed systems. In this thesis, we introduce a state space reduction technique which allows us to treat challenging case studies such as one provided by Astrium EADS for the launcher Ariane 6. We also present how an extension of the Inverse Method, the Behavioral Cartography, can solve the problem of schedulability, i.e., finding the area in the parametric space in which there exists a scheduler that satisfies all the deadlines. We compare this approach to an analytic method to illustrate the interest of our approach

In the second part of this thesis, we are interested in the control of affine switched systems. These systems are governed by a finite family of affine differential equations. At each time step, a controller can choose which dynamics will govern the system for the next time step. Controlling in this sense can be seen as a scheduling on the order of dynamics the system will have to use. The objective for the controller can be to make the system stay in a given area of the state space (stability) or to reach a given region of the state space (reachability). In this thesis, we propose a novel approach that computes a scheduler where the strategy is uniform for dense subsets of the state space. Moreover, our approach only uses forward computation, which is better suited than backward computation for contractive systems. We show that our designed controllers,

systems evolve to a limit cyclic behavior. We apply our method to several case studies from the literature and on a real-life prototype of a multilevel voltage converter. Moreover, we show that our approach can be extended to systems with perturbations and non-linear dynamics.

Résumé

Dans cette thèse, nous nous intéressons au calcul d'ordonnanceurs pour les systèmes hybrides. En fait, nous considérons deux sous-classes des systèmes hybrides, les systèmes temps-réels oú des tâches doivent se partager l'accès à une ressource commune, et les systèmes à commutations où un choix doit être fait sur les dynamiques à choisir en fonction d'objectifs à atteindre.

de permettre l'exécution de toutes ces tâches dans un délai imparti. Dans la première partie de cette thèse, nous nous intéressons aux problèmes d'ordonnancement et prenons comme étude de cas l'ordonnancement de tâches périodiques sur des architectures multiprocesseurs. Nous nous intéressons plus particulièrement à déterminer si l'on peut modifier certaines valeur des paramètres du système tout en respectant les contraintes temporelles sans changer d'ordonnanceur. La méthode inverse permet de prouver de manière formelle la robustesse des systèmes temporisés paramétriques. Nous introduisons une méthode de réduction du nombre d'états nécessaire à la vérification. Cette réduction nous permet de traîter des études de cas intéressantes telle que celle proposée par Astrium EADS pour le lanceur Ariane 6. Nous montrons également comment la Cartographie Comportementale, une extension de la méthode inverse, permet de trouver la zone de l'espace des paramètres où l'on a l'existence d'un ordonnancement satisfaisant les contraintes temporelles. Nous comparons cette approche avec une méthode analytique pour montrer l'intérêt de notre approche.

Dans la seconde partie de cette thèse, nous nous intéressons au contrôle de systèmes affines à commutation. Ces systèmes sont gouvernés par une famille d'équations différentielles linéaires et le contrôleur peut choisir laquelle va gouverner le système pendant le prochain pas de temps. Dans ce cadre, le contrôle peut être vu comme l'ordonnancement des dynamiques que le système va prendre. Le choix de la dynamique peut se faire pour des objectifs de stabilité ou d'accessibilité. Nous proposons une nouvelle méthode qui calcule un contrôleur dont la stratégie est la même pour des ensembles denses de points. Notre méthode utilise le calcul en avant, souvent préférable au calcul à rebours pour les systèmes contractants. Nous montrons que, sous certaines

conditions, le système contrôlé évolue vers un comportement limite. Nous appliquons notre méthode sur plusieurs études de cas issues de la littérature ainsi qu'un exemple réel, un prototype de convertisseur de tension multiniveaux. Enfin, nous montrons que notre méthode s'étend aux systèmes comportant des perturbations ainsi qu'aux systèmes non linéaires.

Remerciements

Tout d'abord, je remercie mon directeur de thèse, Laurent Fribourg, pour ses conseils, son encadrement, son implication dans mon travail, son soutien et pour tous les différents projets de recherche dans lesquels j'ai eu la chance d'être impliqué.

Un grand merci à Éric Goubault et Franck Cassez pour m'avoir fait l'honneur d'être les relecteurs de cette thèse. Merci également à Luc Jaulin et Antoine Girard pour me faire le plaisir de participer au jury de ma thèse. Un très grand merci à Étienne André qui aura été pour cette thèse bien plus qu'un examinateur. Ces contributions envers ce travail sont bien trop nombreuses pour être intégralement listées ici, mais je tiens à lui exprimer ma plus sincère gratitude.

Merci à toutes les personnes avec qui j'ai eu le plaisir de travailler grâce à l'institut Farman : Ludovic Chamoin, Gilles Feld, Éric Florentin, Guillaume Hérault, Pierre-Yves Joubert, Denis Labrousse, Stéphane Lefebvre, David Lesens, Pierre Moro, Bertrand Revol, Christian Rey, Éric Vourc'h, Florian de Vuyst et toutes les autres personnes qui ont collaboré dans ces projets.

I would like to thank Giuseppe Lipari and Youcheng Sun for all our very interesting discussions on schedulability, time analysis and how to improve IMITATOR. I also want to thank Ulrich Kühne for all his help in the development of MINIMATOR and many other things.

Un grand merci également à tous les membres du LSV pour leur gentillesse et tous les bons moments passés ensemble. Je pense en particulier à l'équipe de mots-croisés du midi sans qui les pauses déjeuners auraient été bien moins distrayantes. Je remercie également Dietmar Berwanger pour sa gentillesse et son attention. I would like to give special thanks to Mahsa for being such an awesome office mate and for having made me discover Iranian cooking. Merci également à Sameh pour les discussions enrichissantes que l'on a eues durant les quelques semaines passées ensemble.

Bien entendu, je remercie très chaleureusement mes grands-parents, mes parents, mon frère et plus largement toute ma famille pour m'avoir toujours soutenu dans mes études et dans la vie en général. J'ai une pensée toute particulière pour ceux qui ne sont plus parmi nous.

Je souhaite également remercier tous mes amis, particulièrement Damien, Jérémy, Nicolas et Matthieu pour les longues soirées jeux-vidéos, les parties de golf et tous les weekends partagés.

Et enfin, un immense merci à Adeline pour son soutien, son aide et tout l'amour qu'elle m'a apportés durant ces années de thèse.

Contents

List of Algorithms

List of Figures

List of Tables

General Introduction

General Introduction

Nowadays, computer systems are used everywhere and are growing more and more complex. Supervision of such systems needs to be done automatically and in a sound manner. However, the complexity of these systems is making it hard to both compute a supervisor and to prove its correctness.

One example of those systems are the multiprocessor architectures that are embedded in more and more devices. Such systems can be extremely efficient if the pieces of software running on them are scheduled properly to avoid waiting time and overall delays in the computation. Designing a scheduler for such systems can be hard and proving robustness to minor changes in some timing constants is sought after. Indeed, critical systems such as avionics are not allowed to use such architecture as certifying them cannot be done in an automated manner. In this thesis, we will consider Parametric Timed Automata (PTA) to model those systems and we will show that the Inverse Method can be used to prove the schedulability, exhibit schedulers and give a robustness criteria on the scheduler. We will prove the interest of our approach on several case studies, including one industrial case study provided by Astrium Space Transportation.

Another example is the rise of complexity in power electronics and the need to design schedulers to supervise such systems. In those systems, choices have to be made on the position of switches in order to fulfill some objectives. In general these objectives are to have the system working properly while maintaining some internal values (such as voltage and currents) to parameter valuations where the system cannot be damaged. The high number of possible choice for the switches positions forces to automatically design schedulers for the switches positions. Such power electronic systems can be highly critical. One example of the use of one of those is the voltage converter that allows electricity to go trough the Pyrenees mountains between France and Spain. In this thesis, we will introduce a method to design such schedulers (called controllers in this framework). We will illustrate the interest of our approach on several case studies from the literature and on one real-life prototype of a power electronic devices.

Outline of the Thesis In Part I, we will study the scheduling problem on multiprocessor architectures. Chapter 1 will give an introduction of the problem studied in this part of the thesis. Chapter 2 will be used to recall notions on the Inverse Method and its extension that will be used in this part of the thesis. In Chapter 3, we will introduce a state space reduction technique that is needed to be able to scale up our analysis to be able to treat such problems. In Chapter 4, we will present how we model such problems with PTA and the results of our approach on case studies from the literature and from Astrium Space Transportation. We will conclude this part in Chapter 5

In Part II, we will study control synthesis. Chapter 6 will give an introduction the problem studied in this part of the thesis. Chapter 7 will recall useful concepts of control theory that will be needed for the rest of this part. Chapter 8 will present our approach for the control synthesis and give theoretical results on the correctness of our method, the convergence of the system towards cyclic behavior. We will also present extensions of our method to reachability control, sensitivity and robustness analysis and nonlinear systems. Chapter 9 will present the results of our approach on a real-life prototype of a multi-level converter. We will conclude this part in Chapter 10.

We will give an overall conclusion and future research points in the end.

Contributions In this thesis our contributions are: the proof of the finiteness of the number of tiles in the behavioral cartography of Chapter 2 [Sou10]; the definition of a novel and efficient state merging technique, the correctness theorems and the experiments of the state merging technique of Chapter 3 [AFKS12, AFS13]; the application of the Inverse Method to scheduling problems modeled by stopwatch automata, the full analysis and modeling of the Astrium case study [FLMS12] and the comparison between our approach and analytic method [SSL+13a] of Chapter 4; the decomposition procedure, the theoretical results on the decomposition procedure, the case studies using the decomposition procedure [FS13c] and its extension to other classes of problems [FKS13, FS13b] of Chapter 8 [FS13a]; the modeling of a real-life prototype of multilevel converter and the computation of a controller using the decomposition procedure [SHL+13] in Chapter 9.

Joint work All of this work is a collaboration with Laurent Fribourg (LSV, École Normale Supérieure de Cachan). Étienne André (LIPN, Université Paris 13) collaborated in the work of Part I. Giuseppe Lipari and Youcheng Sun (RETIS lab, Scuola Superiore Sant'Anna) collaborated in the work of Chapter 4, David Lesens and Pierre Moro (Data Management & Software, Astrium Space Transportation) collaborated in the modeling of the Astrium case study of this chap-

ter. Ulrich Kühne (Department of Computer Sciences, University of Bremen) collaborated in the implementation of our tool Minimator and more precisely in the finding of limit cycles of Chapter 8. Work of Chapter 9 has been done in collaboration with Gilles Feld (Département Électronique, Électrotechnique, Automatique), Denis Labrousse, Stéphane Lefebvre and Bertrand Revol (Laboratoire des Systèmes et Applications des Technologies de l'Information et de l'Énergie, ENS Cachan).

Part I

Schedulability Analysis Using The Inverse Method

Chapter 1

Introduction

Schedulability for Central Processing Units (CPUs) deals with assigning resources to tasks (piece of software) so that computing can be done before a given deadline. Relations exist between all the different pieces of software that need to run and usually only one task can be assigned to a given resource at a given time. A system is said to be schedulable if there exists a way to make all the computations before a given deadline.

In this regard, many theoretical results exist for single processor architecture. Unfortunately, they do not extend to multicore, multi processors (MCMP). Worst-case-scenario analysis does not work as well due to the difficulty to exhibit the real worst case scenario for such an architecture. In the industry, schedulers for MCMP architecture are mostly done by hand and then tested. As proposed in [AAM06], one possible solution to automatize this process is to model-check CPU architecture using Timed Automata (TA). This exhaustive analysis ensures that all the cases are checked and properties such as schedulability or mutual exclusion, can be verified using model-checkers. Our main objective in this first part of this thesis is to study MCMP architectures provided by Astrium EADS.

When designing the system, it often happens that constants, such as Worst Case Execution Times (WCETs) are later on modified. Consequently, earlier designed schedulers have to be modified and then retested. An interesting approach would be to provide the scheduler with a zone of validity stating that in this zone, there is no need to retest as the scheduler is still valid and all the properties are still verified.

We will show that this problem can be solved by using the Inverse Method [ACEF09, AS13]. This *inverse method* supposes that we are given a parametric timed automaton $\mathscr{A}$ and a reference valuation π_0 of the parameters that one wants to generalize. It synthesizes a constraint K_0 on the parameters that corresponds to a set such that, for all valuations π of parameters in this set, the

trace sets of $\mathscr{A}[\pi_0]$ and $\mathscr{A}[\pi]$ are equal, i.e., the behavior of the timed automaton $\mathscr{A}[\pi]$ is *(time-abstract) equivalent* to the behavior of $\mathscr{A}[\pi_0]$. This method has two main advantages. First, it gives a criterion of *robustness* by ensuring the correctness of the system for other values for the parameters around the reference valuation. This is of interest when implementing a system: indeed, the exact model with (for example) integer values for timing delays that has been formally verified will necessarily be implemented using values which will not be exactly the ones that have been verified. Second, it allows the system designer to *optimize* some parameters without changing the overall functional behavior of the system, such as schedulability. However, the inverse method suffers from two limitations. First, the constraint synthesized by the method is not necessarily maximal, i.e., there may exist parameter valuations outside the constraint such that the behavior is the same as under the reference valuation. Second, the method focuses on the equality of trace sets, which can be seen as a rather strong property, because the good behavior of a timed system can correspond to different trace sets. We present an approach for solving this problem, based on the inverse method.

A good behavior for systems that we are interested in is schedulability. Given one or more parameters, schedulability consists in finding in which region of the parameter space, the system is schedulable. This can be done using the Behavioral Cartography (BC) [AF10, AS13].

By iterating the inverse method on the integer points of a rectangular parametric domain V_0, we are able to decompose the parametric space into *behavioral tiles*, i.e., parameter zones with a uniform time-abstract behavior. Then, according to a property on traces that one wants to check, it is easy to partition the parametric space into a subset of "good" tiles (which correspond to "good behaviors") and a subset of "bad" ones. This gives a *behavioral cartography* of the system.

Often in practice, what is covered by the behavioral cartography algorithm is not the *bounded* and *integer* subspace of the parameter rectangle V_0, but two major extensions: first, not only the integer points but all the *real-valued* points of the rectangle are covered by the tiles; second, the tiles are often unbounded and cover most of the parametric space *beyond* V_0. Although the cartography may contain holes, i.e., zones not covered by the algorithm, we give sufficient condition for the full coverage of the real-valued bounded parameter domain.

A major interest is that this behavioral cartography does not depend on the property one wants to verify: only the partition into good and bad tiles does. As a consequence, when verifying other properties, it is sufficient to check the property for only one point in each tile in order to get the new partition.

However, the Inverse Method and hence, the Behavioral Cartography, suffers from the state space-explosion when trying to solve those problems. Tech-

niques exist for merging states when the no parameters are used in the model. In this thesis, we propose a technique for merging in the parametric case.

Outline of this part. In Chapter 2, we recall all the definitions and algorithms of the Inverse Method and the Behavioral Cartography. In Chapter 3, we present our state reduction technique for PTAs. In Chapter 4, we present our results on scheduling case study including one proposed by Astrium EADS.

Chapter 2

The Inverse Method for Parametric Timed Automata

In this chapter, we present the formalisms used throughout this part of the thesis.

In particular, we present timed automata [AD94], a powerful modeling formalism for real-time systems. Since we focus on synthesizing values for timing parameters of a system guaranteeing a good behavior, we will also use a parametric extension of timed automata, namely *parametric timed automata* [AHV93]. This chapter presents their syntax and semantics, and more generally all the necessary formalisms to understand the rest of this thesis.

In this chapter, we consider the following inverse problem:

> **"Given a reference parameter valuation, synthesize a constraint on the parameters such that, for any valuation satisfying this constraint, the trace set of the system is the same as under the reference valuation".**

This notion of equality of trace sets gives a guarantee of time-abstract equivalence of the behavior of the system. We recall a method solving the inverse problem, the *Inverse Method*.

The material of this chapter is mostly borrowed from [AS13].

Outline of the Chapter

We describe clocks, parameters, and constraints on the clocks and parameters in Section 2.1 and labeled transition systems in Section 2.2. We then introduce the syntax and semantics of timed automata in Section 2.3, and parametric timed automata in Section 2.4. We recall the formal definition of the inverse problem in Section 2.5 by using the example of "flip-flop" asynchronous cir-

cuit.. We then introduce the inverse method in Section 2.6, give results of correctness and termination, and state properties of the method.

2.1 Constraints on Clocks and Parameters

2.1.1 Clocks

Throughout the first part of this thesis, we assume a fixed set $X = \{x_1, \ldots, x_H\}$ of *clocks*. A *clock* is a variable x_i with value in $\mathbb{R}_+$, which denotes the set of non-negative real numbers. All clocks evolve linearly at the same rate. We define a *clock valuation* as a function $w : X \to \mathbb{R}_+$ assigning a non-negative real value to each clock variable. We will often identify a valuation w with the point $(w(x_1), \ldots, w(x_H))$. Given a constant $d \in \mathbb{R}_+$, we use $X + d$ to denote the set $\{x_1 + d, \ldots, x_H + d\}$. Similarly, we write $w + d$ to denote the valuation such that $(w + d)(x) = w(x) + d$ for all $x \in X$.

2.1.2 Parameters

We assume a fixed set $P = \{p_1, \ldots, p_M\}$ of *parameters*, i.e., unknown constants. A *parameter valuation* π is a function $\pi : P \to \mathbb{R}_+$ assigning a nonnegative real value to each parameter. There is a one-to-one correspondence between valuations and points in $(\mathbb{R}_+)^M$. We will often identify a valuation π with the point $(\pi(p_1), \ldots, \pi(p_M))$.

2.1.3 Constraints

We define here constraints as a set of linear inequalities.

Syntax of Constraints

Definition 1. *Let V be a set of variables of the form $V = \{v_1, \ldots, v_N\}$. A* linear inequality *on the variables of V is an inequality $e \prec e'$, where $\prec \in \{<, \leq\}$, and e, e' are two linear terms of the form*

$$\sum_{1 \leq i \leq N} \alpha_i v_i + d$$

where $v_i \in V$, $\alpha_i \in \mathbb{R}_+$, for $1 \leq i \leq N$, and $d \in \mathbb{R}_+$.

Note that we define the coefficients of the linear inequalities as *positive reals*. It would be equivalent to define them as *positive rationals*, since we consider only linear inequalities. Both definitions are found in the literature; we

suppose here that, since we are addressing the problem of the verification of *real*-time systems, we consider *real* valued constants.

We assume in the following that all inequalities are linear, and we will simply refer to linear inequalities as *inequalities*.

Definition 2. *Let V be a set of variables of the form $V = \{v_1, \ldots, v_N\}$. Given an inequality J on the variables of V of the form $e < e'$ (respectively $e \leq e'$), the* negation *of J, denoted by $\neg J$, is the linear inequality $e' \leq e$ (respectively $e' < e$).*

Definition 3. *Let V be a set of variables of the form $V = \{v_1, \ldots, v_N\}$. A* convex linear constraint on the variables of V *is a conjunction of inequalities on the variables of V.*

We assume in the following that all constraints are both convex and linear, and we will simply refer to convex linear constraints as *constraints*.

Definition 4. *An* inequality on the clocks *is an inequality on the set of clocks X. A* constraint on the clocks *is a constraint on the set of clocks X.*

Definition 5. *An* inequality on the parameters *is an inequality on the set of parameters P. A* constraint on the parameters *is a constraint on the set of parameters P.*

Definition 6. *An* inequality on the clocks and the parameters *is an inequality on $X \cup P$. A* constraint on the clocks and the parameters *is a constraint on $X \cup P$.*

We denote by $\mathscr{L}(X)$ the set of all constraints on the clocks, by $\mathscr{L}(P)$ the set of all constraints on the parameters, and by $\mathscr{L}(X \cup P)$ the set of all constraints on the clocks and the parameters.

In the sequel, the letter J will denote an inequality on the parameters, the letter D will denote a constraint on the clocks, the letter K will denote a constraint on the parameters, and the letter C will denote a constraint on the clocks and the parameters.

Semantics of Constraints

Given a constraint D on the clocks and a clock valuation w, $D[w]$ denotes the expression obtained by replacing each clock x in D with $w(x)$. A clock valuation w *satisfies* constraint D (denoted by $w \models D$) if $D[w]$ evaluates to true.

Given a parameter valuation π and a constraint C on the clocks and the parameters, $C[\pi]$ denotes the constraint on the clocks obtained by replacing each parameter p in C with $\pi(p)$. Likewise, given a clock valuation w, $C[\pi][w]$ denotes the expression obtained by replacing each clock x in $C[\pi]$ with $w(x)$.

We say that a parameter valuation π *satisfies* a constraint C, denoted by $\pi \models C$, if the set of clock valuations that satisfy $C[\pi]$ is nonempty. We use the notation $<w,\pi> \models C$ to indicate that $C[\pi][w]$ evaluates to true.

A convex linear constraint on the clocks and the parameters can also be interpreted as a set of points in the space $\mathbb{R}^{M+H}$, more precisely as a convex polyhedron. We will use these notions synonymously. In this geometric context, a valuation w satisfying a constraint C is equivalent to the polyhedron C containing the corresponding point w, written as $w \in C$. For a partial valuation w (i.e., a point of a subspace of C), we write $w \in C$ if and only if w is contained in the projection of C on the variables of w.

Given two constraints C_1 and C_2 on the clocks and the parameters, we say that C_1 is *included in* C_2, denoted by $C_1 \subseteq C_2$, if $\forall w,\pi : <w,\pi> \models C_1 \Rightarrow <w,\pi> \models C_2$. We have that $C_1 = C_2$ if and only if $C_1 \subseteq C_2$ and $C_2 \subseteq C_1$.

Similarly to the semantics of constraints on the clocks and the parameters, we say that a parameter valuation π *satisfies* a constraint K on the parameters, denoted by $\pi \models K$, if the expression obtained by replacing each parameter p in K with $\pi(p)$ evaluates to true. Given two constraints K_1 and K_2 on the parameters, we say that K_1 is *included in* K_2, denoted by $K_1 \subseteq K_2$, if $\forall \pi : \pi \models K_1 \Rightarrow \pi \models K_2$. We have that $K_1 = K_2$ if and only if $K_1 \subseteq K_2$ and $K_2 \subseteq K_1$. We will consider `true` as a constraint on the parameters, corresponding to the set of all possible values for P.

Given a constraint C on the clocks and the parameters, we denote by $C{\downarrow_P}$ the constraint on the parameters obtained by projecting C onto the set of parameters, that is after elimination of the clock variables. Formally, $C{\downarrow_P} = \{\pi \mid \exists w : <w,\pi> \models C\}$.

Sometimes we will refer to a variable domain X', which is obtained by renaming the variables in X. Explicit renaming of variables is denoted by the substitution operation. Given a constraint C on the clocks and the parameters, we denote by $C_{[X \leftarrow X']}$ the constraint obtained by replacing in C the variables of X by the variables of X'.

We define the *time elapsing* of C, denoted by $C{\uparrow}$, as the constraint over X and P obtained from C by delaying an arbitrary amount of time. Note that, of course, only clocks can evolve; parameters are unknown constants and therefore remain constant. Formally:

$$C{\uparrow} = \left((C \wedge X' = X + d){\downarrow_{X' \cup P}}\right)_{[X' \leftarrow X]}$$

where d is a new parameter with values in $\mathbb{R}_+$, and X' is a renamed set of clocks. The inner part of the expression adds the same delay d to all clocks; the projection onto $X' \cup P$ eliminates the original set of clocks X, as well as the variable d; the outer part of the expression renames clocks X' with X.

2.2 Labeled Transition Systems

We introduce below labeled transition systems, which will be used later in this section to represent the semantics of timed automata.

Definition 7. *A* labeled transition system *over a set of symbols* Σ *is a triple* $\mathrm{L} = (S, S_0, \Rightarrow)$*, with* S *a set of* states*,* $S_0 \subset S$ *a set of* initial states*, and* $\Rightarrow \in S \times \Sigma \times S$ *a* transition relation*. We write* $s \overset{a}{\Rightarrow} s'$ *for* $(s, a, s') \in \Rightarrow$*. A* run *(of length* m*) of* L *is a finite alternating sequence of states* $s_i \in S$ *and symbols* $a_i \in \Sigma$ *of the form* $s_0 \overset{a_0}{\Rightarrow} s_1 \overset{a_1}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} s_m$*, where* $s_0 \in S_0$*. A state* s_i *is* reachable *if it belongs to some run* r*.*

2.3 Timed Automata

Timed automata are an extension of standard finite-state automata allowing the use of clocks, that is real-valued variables increasing linearly at the same rate. Such clocks can be compared with constants in constraints that allow (or not) to stay in a location ("invariants") or to take a transition ("guards"). At each transition, it is possible to reset some of the clocks of the system. This formalism allows the parallel composition of several timed automata, which behave like a single one, and thus provides the designer with a powerful and intuitive way to represent timed systems. It is important to note that the model of timed automata is very sensitive to the size of the automata and the number of automata in parallel, thus often leading to the state-space explosion problem. However, powerful tools, such as UPPAAL [LPY97], or Kronos [Yov97], have been implemented, allowing designers to model and verify efficiently timed systems modeled by timed automata.

2.3.1 Syntax

Definition 8. *A* timed automaton $\mathscr{A}$ *is a* 6*-tuple of the form* $\mathscr{A} = (\Sigma, Q, q_0, X, I, \rightarrow)$*, where*

- Σ *is a finite set of* actions*,*
- Q *is a finite set of* locations*,*
- $q_0 \in Q$ *is the* initial location*,*
- X *is a set of* clocks*,*

- $I : Q \to \mathscr{L}(X)$ *is the* invariant*, assigning to every* $q \in Q$ *a constraint* $I(q)$ *on the clocks, and*

- $\to$ *is a transition relation consisting of elements of the form* (q, g, a, ρ, q')*, also denoted by* $q \overset{g,a,\rho}{\to} q'$*, where* $q, q' \in Q$*,* $a \in \Sigma$*,* $\rho \subseteq X$ *is a set of clock variables to be* reset *by the transition, and* $g \in \mathscr{L}(X)$ *is the* guard *of the transition.*

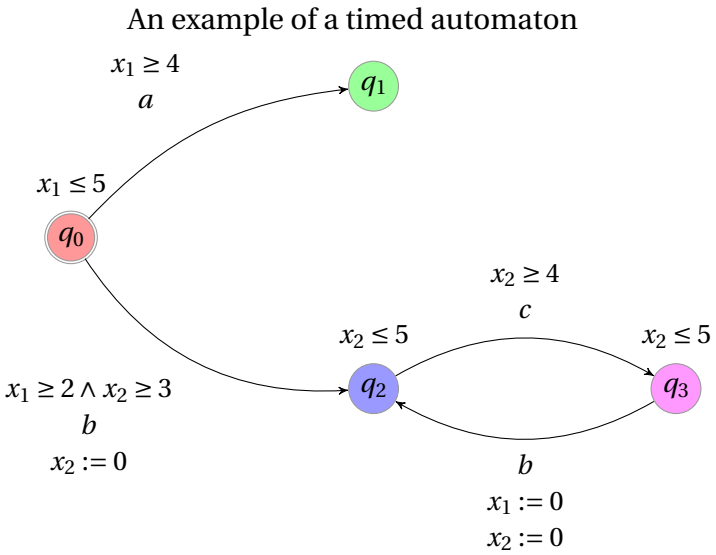
Note that we use a more permissive definition of the constraints used in guards and invariants than in the original definition of timed automata (see [AD94]). Indeed, we allow the use of conjunctions of any linear inequalities on the clocks, whereas the original definition usually considers conjunctions of comparisons of a single clock with a constant. This more permissive definition has usually an impact on the decidability (the addition of clock values within a constraint leads to undecidability [AD94]), but this has no impact in this work, mainly because of the use of *parametric* timed automata, where the parameters bring themselves undecidability in the general case. Furthermore, many tools for (parametric) timed automata allow more permissive definitions than the original one.

Timed automata are often extended in practice with *discrete variables*, which can be used in guards and transitions, updated within the transitions, and sometimes even used as a factor for clocks. However, in most cases, they represent only syntactic sugar for the discrete space (i.e., locations). As a consequence, we will not use them in any theoretical part of this thesis. Note nevertheless that many tools for (parametric) timed automata allow the use of such discrete variables. Some of the case studies contained hereafter also use them.

The graphical representation of a timed automaton $\mathscr{A}$ is an oriented graph where vertices correspond to locations, and edges correspond to actions of $\mathscr{A}$. We follow the following conventions for the graphical representation of timed automata: locations are represented by nodes, above of which the invariant of the location is written; transitions are represented by arcs from one location to another location, labeled by the associated guard, the action name and the set of clocks to be reset (guards and invariants equal to `true` are omitted). The initial location is here represented using a double circle.

Example 1. *We give in Figure 1 an example of a timed automaton containing 4 locations (*q_0*,* q_1*,* q_2 *and* q_3*), 3 actions (*a*,* b *and* c*) and 2 clocks (*x_1 *and* x_2*). The initial location is* q_0*.*

In this timed automaton, q_0 *has invariant* $x_1 \leq 5$*,* q_1 *has invariant* `true`*, and both* q_2 *and* q_3 *have invariant* $x_2 \leq 5$*. The transition from* q_0 *to* q_1 *has guard* $x_1 \geq 4$ *through action* a*; no clock is reset. The transition from* q_0 *to* q_2 *has guard*



$x_1 \geq 2 \wedge x_2 \geq 3$ *through action* b*, and resets clock* x_2*. The transitions between* q_2 *and* q_3 *can be explained similarly.*

2.3.2 Semantics

The semantics of timed automata is given under the form of a labeled transition system, where states are pairs made by a location and a valuation for each clock.

Definition 9. *Let* $\mathscr{A} = (\Sigma, Q, q_0, X, I, \rightarrow)$ *be a timed automaton. The* concrete semantics of $\mathscr{A}$ *is the labeled transition system* $(S, S_0, \Rightarrow)$ *over* Σ *where*

$$S = \{(q, w) \in Q \times (X \rightarrow \mathbb{R}_+) \mid w \models I(q)\},$$
$$S_0 = \{(q_0, w) \mid w \models I(q_0) \wedge w = (w_0, \ldots, w_0) \text{ for some } w_0 \in \mathbb{R}_+\}$$

and the transition predicate $\Rightarrow$ *is specified by the following three rules. For all* $(q, w), (q', w') \in S, d \geq 0$ *and* $a \in \Sigma$*,*

- $(q, w) \xrightarrow{a} (q', w')$ *if* $\exists g, \rho : q \overset{g,a,\rho}{\rightarrow} q'$ *and* $w \models g$ *and* $w' = \rho(w)$*;*
- $(q, w) \xrightarrow{d} (q', w')$ *if* $q' = q$ *and* $w' = w + d$*;*
- $(q, w) \overset{a}{\Rightarrow} (q', w')$ *if* $\exists d, w'' : (q, w) \xrightarrow{a} (q', w'') \xrightarrow{d} (q', w')$*.*

We consider with the definition of S_0 that all clocks are initially set to 0, or have evolved linearly in the bounds given by $I(q_0)$. A state (respectively run) in the concrete semantics is a state (respectively run) of the corresponding labeled transition system, and will be referred to as a *concrete state* (respectively *concrete run*).

A concrete run is represented under the form of a branch where states are depicted within nodes containing the name of the location and the value of each of the clocks, and transitions are depicted using edges labeled with the name of the action.

Example 2. *Consider again the timed automaton $\mathscr{A}$ of Example 1. Then Figure 2.1 depicts an example of a concrete run for $\mathscr{A}$.*

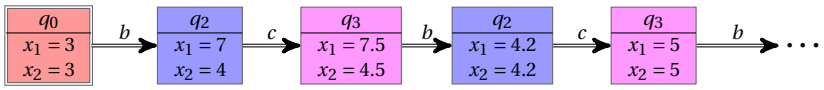


Figure 2.1: Example of a concrete run for a timed automaton

This run is obtained as follows: one starts from the initial location q_0 where both clocks have evolved during 3 time units. Then, we take action b, reset x_2 and spend 4 time units in q_2. Then, we take action c, and spend 0.5 time unit in q_3. Then, we take action b, reset both clocks and spend 4.2 time units in q_2. Then, we take action c, and spend 0.8 time units in q_3, and so on.

The power of timed automata relies in the fact that one can construct a finite partition of the infinite space of clock valuations. In particular, this construction is suitable to perform reachability analysis. The main theoretical advantage of timed automata relies in its decidability results. In particular, it has been shown that the reachability of a state is decidable. Moreover, various *timed* temporal logics (e.g., [ACD93]) have been designed, and various decidability results have been shown (e.g., [ACD93, HRSV02, WY03, Fin06, BBBB09]).

Traces

We now introduce the notion of *trace*, that abstracts part of a system's behavior. In the literature, one usually considers either a state-based approach or an action-based approach (see, e.g., [BK08]). We consider here a combined state- and action-based approach: a trace is an alternating sequence of locations and actions. Note that, for a deterministic timed automaton, that is a timed automaton such that there is at most one transition leaving a given location with a given action, there is an equivalence between the state-based, the action-based and the combined approaches (because of the unicity of the initial location). This can be the case for hardware verification when one models circuits at the gate level. Indeed, when one models each gate of the circuit with a different timed automaton, where each location corresponds to a different value of the input and output signals of the gate, and each transition corresponds to a rise

or a fall of a signal of the global system, then the composition of the timed automata modeling each gate is deterministic. In that case, if one considers a sequence of locations, it is possible to retrieve the corresponding sequence of actions (from a given initial location), and conversely.

We define more formally the notion of trace in the following definition.

Definition 10. *Given a timed automaton $\mathscr{A}$ and a concrete run r of $\mathscr{A}$ of the form $(q_0, w_0) \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} (q_m, w_m)$, the* trace associated with r *is the alternating sequence of locations and actions $q_0 \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} q_m$. We say that location q_i, for $1 \leq i \leq m$, belongs to the trace.*

A trace is built from a run by removing the valuation of the clocks, and therefore can be seen as a *time-abstract run*. We depict traces under a graphical form using boxed nodes labeled with locations and double arrows labeled with actions.

Example 3. *The trace associated with the concrete run of Example 2 is depicted in Figure 2.2.*

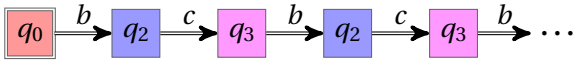


Figure 2.2: Example of a trace associated with a concrete run for a timed automaton

We define below the notion of acyclic trace as a trace which never passes twice by the same location, that is a trace whose locations are all different.

Definition 11. *Given a trace $T = q_0 \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} q_m$, T is said to be an* acylic trace *if:*

$$\forall q_i, q_j, i < j < m, q_i \neq q_j$$

Given two traces, we define the following notion of prefix of a trace.

Definition 12. *Given a trace $T = q_0 \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} q_m$, the* prefix of length n of T *is the trace denoted by $|T|_n$ and defined as follows:*

$$|T|_n = \begin{cases} q_0 \overset{a_0}{\Rightarrow} \cdots \overset{a_{n-1}}{\Rightarrow} q_n & \text{if } n < m \\ q_0 \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} q_m & \text{otherwise} \end{cases}$$

Similarly, we say that a trace T_1 is a prefix *of a trace T_2 if there exists $n \geq 0$ such that $|T_2|_n = T_1$.*

We now define the following notion of trace set.

Definition 13. *Given a timed automaton $\mathscr{A}$, the* trace set of $\mathscr{A}$ *refers to the set of traces associated with the runs of $\mathscr{A}$.*

Often, when depicting trace sets, we will not depict each trace separately, but depict the trace set under the form of a tree or a graph. Note, however, that this graph structure is only used for the sake of simplicity of representation of the possible traces, and does not contain any information on the possible *branching* behavior of the system.

Example 4. *The trace set associated with the timed automaton of Example 1 is depicted in Figure 2.3.*

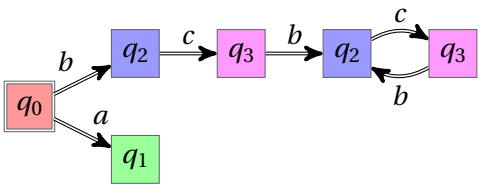


Figure 2.3: Example of a trace set of a timed automaton

This trace set contains an infinite number of finite traces. Note also that it is obviously not acyclic, because there are (actually infinitely many) traces passing several times by locations q_2 and q_3.

We extend the notion of acyclicity of a trace to trace sets, and say that a trace set is *acyclic* if all its traces are acyclic. We also say that a location q belongs to the trace set of $\mathscr{A}$ if it belongs to a trace of the trace set of $\mathscr{A}$.

In the following, we are interested in verifying properties on the trace set of $\mathscr{A}$. For example, given a predefined set of "bad locations", a reachability property is satisfied by a trace if this trace never contains a bad location; such a trace is "good" with respect to this reachability property. A trace can also be said to be "good" if a given action always occurs before another one within the trace (see example in Section 2.5.1). Actually, the good behaviors that can be captured with trace sets are relevant to *linear-time properties* [BK08], which can express properties more general than reachability properties.

Definition 14. *Given a timed automaton $\mathscr{A}$, and a property on traces, we say that a trace of $\mathscr{A}$ is* good *if it satisfies the property, and* bad *otherwise. Likewise, we say that the trace set of $\mathscr{A}$ is* good *if all its traces are good, and bad otherwise.*

2.4 Parametric Timed Automata

Parametric timed automata are an extension of the class of timed automata to the parametric case. Parametric timed automata allow within guards and invariants the use of parameters in place of constants [AHV93]. This model is interesting when one does not only want to check that a system is correct for *one* value of the constants, but for a whole dense set of values. The model of parametric timed automata is also interesting to synthesize parameters for which a given property is satisfied.

Unfortunately, for most interesting problems, parametric timed automata lose the decidability results proved for timed automata. In particular, the reachability of a state is not decidable (although semi-algorithms do exist, i.e., if the algorithm terminates, then the result is correct). Moreover, parametric timed automata are even more sensitive to the state space explosion problem, because of the addition of the parameters. In practice, this comes also from the fact that the data structures used to represent parametric timed automata are far less efficient than the ones used for timed automata (typically Difference Bound Matrices, proposed in [BM83] for the analysis of time Petri nets, and introduced in [Dil89] for timed automata). Structures allowing to handle parametric timed models include Parametric Difference Bound Matrices (an extension of Difference Bound Matrices, proposed in [HRSV02]), SAT-solvers, SMT-solvers and polyhedra. Avoiding the explosion of the state space, and finding cases for which analyses are decidable for parametric timed automata, are actually some of the motivations for the techniques described in this thesis.

2.4.1 Syntax

Definition 15. *A* parametric timed automaton $\mathscr{A}$ *is a* 8*-tuple of the form* $\mathscr{A} = (\Sigma, Q, q_0, X, P, K, I, \rightarrow)$*, where*

- Σ *is a finite set of* actions*,*
- Q *is a finite set of* locations*,*
- $q_0 \in Q$ *is the* initial location*,*
- X *is a set of* clocks*,*
- P *is a set of* parameters*,*
- K *is an* initial constraint *on the parameters of* P*,*
- $I : Q \rightarrow \mathscr{L}(X \cup P)$ *is the* invariant*, assigning to every* $q \in Q$ *a constraint* $I(q)$ *on the clocks and the parameters, and*

- $\rightarrow$ *is a transition relation consisting of elements of the form* (q, g, a, ρ, q'), *also denoted by* $q \overset{g,a,\rho}{\rightarrow} q'$, *where* $q, q' \in Q$, $a \in \Sigma$, $\rho \subseteq X$ *is a set of clock variables to be* reset *by the transition, and* $g \in \mathscr{L}(X \cup P)$ *is the transition* guard.

The initial constraint K is useful to define constrained models, where some parameters are already related. For example, in a timed model with two parameters *min* and *max*, one may want to constrain *min* to be always smaller or equal to *max*, that is $K = \{min \leq max\}$. Although it does not add expressive power (this constraint could be "simulated" by simply adding it to the invariant of the initial location), it is largely used in practice. All case studies considered here make use of this initial constraint. Furthermore, this constraint K can be refined in order to constrain the model further. In the following, given a parametric timed automaton $\mathscr{A} = (\Sigma, Q, q_0, X, P, K, I, \rightarrow)$, we will often denote this parametric timed automaton by $\mathscr{A}(K)$ when clear from the context, in order to emphasize that only K will change when performing repeated analysis on refined models of $\mathscr{A}$.

We make use for parametric timed automata of the same graphical representation as for timed automata, that is an oriented graph where vertices correspond to the locations, and edges correspond to the actions. The graphical representation of a parametric timed automaton will be referred to as its *associated graph*.

Example 5. *We give in Figure 2.4 an example of a parametric timed automaton containing 4 locations* (q_0, q_1, q_2 *and* q_3), *3 actions* (a, b *and* c), *2 clocks* (x_1 *and* x_2) *and 2 parameters* (p_1 *and* p_2). *The initial location is* q_0.

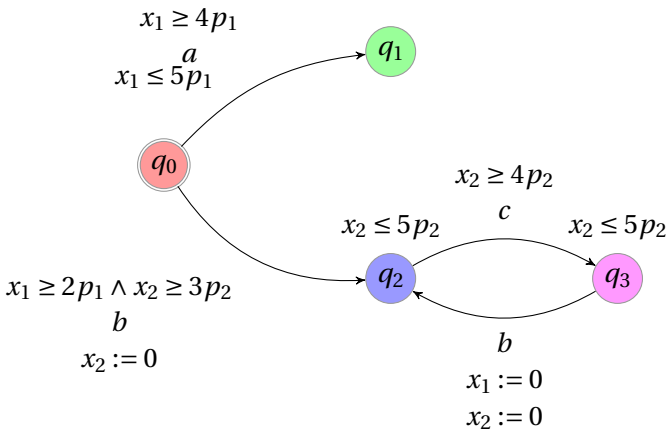


Figure 2.4: An example of a parametric timed automaton

In this parametric timed automaton, q_0 has invariant $x_1 \leq 5p_1$, q_1 has invariant `true`*, and both q_2 and q_3 have invariant $x_2 \leq 5p_2$. The transition from q_0 to q_1 has guard $x_1 \geq 4p_1$ through action a; no clock is reset. The transition from q_0 to q_2 has guard $x_1 \geq 2p_1 \wedge x_2 \geq 3p_2$ through action b, and resets clock x_2. The transitions between q_2 and q_3 can be explained similarly.*

Instantiation of a Parametric Timed Automaton

Given a parametric timed automaton $\mathscr{A} = (\Sigma, Q, q_0, X, P, K, I, \rightarrow)$ and a parameter valuation $\pi = (\pi(p_1), \ldots, \pi(p_M))$, $\mathscr{A}[\pi]$ denotes the parametric timed automaton $\mathscr{A}(K^\pi)$, where K^π is $K \wedge \bigwedge_{i=1}^{M} p_i = \pi(p_i)$. This corresponds to the parametric timed automaton obtained from $\mathscr{A}$ by substituting every occurrence of a parameter p_i by constant $\pi(p_i)$ in the guards and invariants. We say that p_i is *instantiated* with $\pi(p_i)$. Note that, as all parameters are instantiated, $\mathscr{A}[\pi]$ is a standard timed automaton.

Example 6. *Consider again the parametric timed automaton $\mathscr{A}$ described in Example 5. Also consider the following reference valuation π of the parameters: $\pi : \{p_1 = 1, p_2 = 1\}$. Then, the (non-parametric) timed automaton $\mathscr{A}[\pi]$ is the one described in Example 1.*

We now define the notion of *acyclic* parametric timed automaton. This acyclicity of a parametric timed automaton can be deduced purely syntactically from its graphical representation, that is if this representation is acyclic.

Definition 16. *We say that a parametric timed automaton is* graphically acyclic *(or, more simply,* acyclic*) if its associated graph is acyclic.*

Note that the trace set associated with an acyclic parametric timed automaton is necessarily acyclic. (However, note that if a trace set is acyclic, its parametric timed automaton is not necessarily acyclic.)

Parallel Composition of Parametric Timed Automata

We now introduce the notion of network of parametric timed automata, and show in the following definition how N parametric timed automata can be composed into a single parametric timed automaton, by performing a product of the N automata.

Definition 17. *Let $N \in \mathbb{N}$. For all $1 \leq i \leq N$, let $\mathscr{A}_i = (\Sigma_i, Q_i, (q_0)_i, X_i, P_i, K_i, I_i, \rightarrow_i)$ be a parametric timed automaton. The sets Q_i are mutually disjoint. A* network of parametric timed automata *is $\mathscr{A} = \mathscr{A}_1 \| \ldots \| \mathscr{A}_N$, where $\|$ is the operator for parallel composition defined in the following way.*

This network of parametric timed automata corresponds to the parametric timed automaton $\mathscr{A} = (\Sigma, Q, q_0, X, P, K, I, \rightarrow)$ *where*

- $\Sigma = \bigcup_{i=1}^{N} \Sigma_i$,
- $Q = \Pi_{i=1}^{N} Q_i$,
- $q_0 = \langle (q_0)_1, \ldots, (q_0)_N \rangle$,
- $X = \bigcup_{i=1}^{N} X_i$,
- $P = \bigcup_{i=1}^{N} P_i$,
- $K = \bigwedge_{i=1}^{N} K_i$,
- $I(\langle q_1, \ldots, q_N \rangle) = \bigwedge_{i=1}^{N} I_i(q_i)$ *for all* $\langle q_1, \ldots, q_N \rangle \in Q$,

and $\rightarrow$ *is defined as follows. For all* $a \in \Sigma$, *let* T_a *be the subset of indices* $i \in 1, \ldots, N$ *such that* $a \in \Sigma_i$. *For all* $a \in \Sigma$, *for all* $\langle q_1, \ldots, q_N \rangle \in Q$, *for all* $\langle q'_1, \ldots, q'_N \rangle \in Q$, $(\langle q_1, \ldots, q_N \rangle, g, a, \rho, \langle q'_1, \ldots, q'_N \rangle) \in \rightarrow$ *if:*

- *for all* $i \in T_a$, *there exist* g_i, ρ_i *such that* $(q_i, g_i, a, \rho_i, q'_i) \in \rightarrow_i$, $g = \bigwedge_{i \in T_a} g_i$, $\rho = \bigcup_{i \in T_a} \rho_i$, *and,*
- *for all* $i \notin T_a$, $q'_i = q_i$.

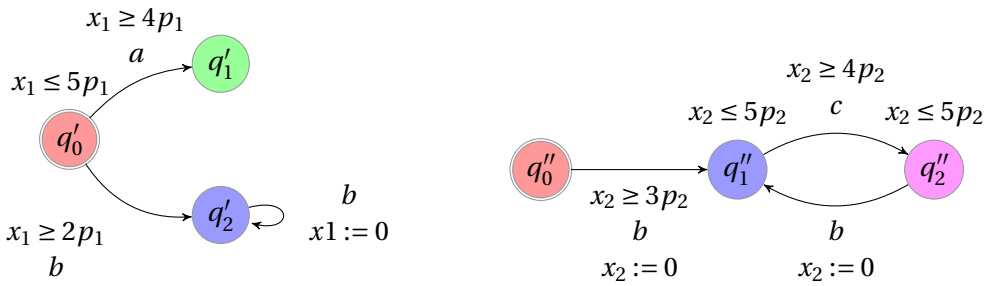
In this definition, the set of actions is the union of the "local" sets of actions (i.e., of each automaton $\mathscr{A}_i$), and similarly for the sets of clocks and parameters. The set of locations of the resulting automaton is the product of the local sets of locations: hence, each location of the resulting automaton is composed of a location of each local automaton. The initial location is made of the initial location of each local automaton. The initial constraint is the intersection of the local initial constraints. Each invariant is the intersection of the local invariants associated to a local location. Finally, a global transition can be taken as follows. For an action a, we compute the set of local automata (denoted by T_a) such that $a \in \Sigma_i$. Then, this transition can be taken if there exists a successor location through a for each automaton in T_a. The guard is obtained by intersecting the local guards of T_a, and the set of clocks to reset is the union of the local sets of clocks to reset. The local location of automata not in T_a remains unchanged.

Sometime, we meet in practice the requirement that the set of clocks and parameters of each of the timed automata in parallel must be mutually disjoint. Here, for the sake of generality, we do allow the shared use of clocks and parameters between different automata.

Note that, in practice, most tools perform an *on-the-fly* (and partial) composition of the product. Indeed, the computation of this product is usually prohibitively expensive. For example, the composition of 10 automata with 10 locations each will result in a global automaton with a set of locations of size 10^{10}. Hence, most tools only compose the state space that is effectively reached, by computing each state when needed only.

Example 7. *We give in Figure 7 an example of a network of 2 parametric timed automata.*

Example of a network of parametric timed automata



(a) $\mathscr{A}'$ (b) $\mathscr{A}''$

The composition of those 2 parametric timed automata in parallel ($\mathscr{A}' \| \mathscr{A}''$) corresponds to the parametric timed automaton from Example 5, where $q_0 = (q'_0, q''_0)$, $q_1 = (q'_1, q''_0)$, $q_2 = (q'_2, q''_1)$ and $q_3 = (q'_2, q''_2)$.

2.4.2 Semantics

We now define the semantics of parametric timed automata. We first introduce the notion of symbolic state.

Definition 18. *Let $\mathscr{A} = (\Sigma, Q, q_0, X, P, K, I, \rightarrow)$ be a parametric timed automaton. A* (symbolic) state *s of $\mathscr{A}(K)$ is a pair (q, C) where $q \in Q$ is a location, and $C \in \mathscr{L}(X \cup P)$ a constraint on the clocks and the parameters.*

For each valuation π of the parameters P, we may view a symbolic state $s = (q, C)$ as the set of pairs (q, w) where w is a clock valuation such that $<w, \pi> \models C$.

We define the inclusion of a state in another one as the equality of locations and inclusion of constraints.

Definition 19. *We say that a state $s_1 = (q_1, C_1)$ is* included *in a state $s_2 = (q_2, C_2)$, denoted by $s_1 \subseteq s_2$, if $q_1 = q_2$ and $C_1 \subseteq C_2$.*

We say that two states $s_1 = (q_1, C_1)$ *and* $s_2 = (q_2, C_2)$ *are* equal*, denoted by* $s_1 = s_2$*, if* $q_1 = q_2$ *and* $C_1 = C_2$*.*

We now define the inclusion of a set S_1 of states in another set S_2. Observe that this notion does not refer to the inclusion of each state of S_1 into a state of S_2, but to the *equality* of each state S_1 with a state S_2. Hence, all states of S_1 exactly appear in S_2.

Definition 20. *We say that a set of states* S_1 *is* included *into a set of states* S_2*, denoted by* $S_1 \sqsubseteq S_2$*, if*

$$\forall s : s \in S_1 \Rightarrow s \in S_2.$$

We say that two sets of states S_1 *and* S_2 *are* equal*, denoted by* $S_1 = S_2$*, if* $S_1 \sqsubseteq S_2$ *and* $S_2 \sqsubseteq S_1$*.*

We are interested in checking whether the constraint associated with a symbolic state is satisfied by a given valuation of the parameters. This refers to the following notion of π-compatibility.

Definition 21. *Let* $\mathscr{A}$ *be a parametric timed automaton, and* $s = (q, C)$ *be a state of* $\mathscr{A}$*. The state* s *is said to be* compatible with π *or, more simply,* π-compatible *if* $\pi \models C$*, and* π-incompatible *otherwise.*

The *initial state* of $\mathscr{A}(K)$ is a symbolic state s_0 of the form (q_0, C_0), where $C_0 = K \wedge I(q_0) \wedge \bigwedge_{i=1}^{H-1} x_i = x_{i+1}$. (Recall that H is the number of clocks.) In this expression, K is the initial constraint on the parameters, $I(q_0)$ is the invariant of the initial state, and the rest of the expression lets clocks evolve from the same initial value.

The semantics of parametric timed automata is given in the following under the form of a labeled transition system.

Definition 22. *Let* $\mathscr{A} = (\Sigma, Q, q_0, X, P, K, I, \rightarrow)$ *be a parametric timed automaton. The* symbolic semantics of $\mathscr{A}$ *is the labeled transition system* $(S, S_0, \Rightarrow)$ *over* Σ *where*

$$S = \{(q, C) \in Q \times \mathscr{L}(X \cup P) \mid C \subseteq I(q)\},$$
$$S_0 = \{(q_0, K \wedge I(q_0) \wedge \textstyle\bigwedge_{i=1}^{H-1} x_i = x_{i+1})\}$$

and a transition $(q, C) \overset{a}{\Rightarrow} (q', C')$ *belongs to* $\Rightarrow$ *if* $\exists C'' : (q, C) \xrightarrow{a} (q', C'') \xrightarrow{d} (q', C')$*, with*

- *discrete transitions* $(q, C) \xrightarrow{a} (q', C')$ *if there exists* $(q, g, a, \rho, q') \in \rightarrow$ *and*

$$C' = \Big(\big(C(X) \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(q')(X')\Big)_{[X' \leftarrow X]}, \text{ and}$$

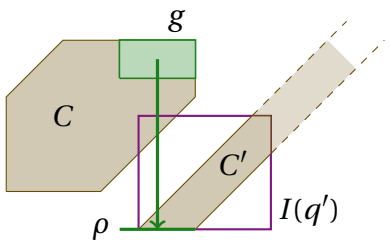


Figure 2.5: Forward reachability for timed automata

- *delay transitions* $(q,C) \xrightarrow{d} (q,C')$ *with* $C' = C{\uparrow} \wedge I(q)(X)$.

Let us explain this definition. A transition in the symbolic semantics is the combination of a discrete transition followed by a delay transition. A discrete transition can be taken as follows: first, the original constraint C is intersected with the guard g. We use notation $g(X)$ to denote that the set of variables is X; similarly, $I(q)(X')$ is used to denote that the set of variables is X'. Then, the reset operation is performed by $X' = \rho(X)$; we use $X' = \rho(X)$ to denote that all clocks of X' are equal to clocks of X, except clocks belonging to the set ρ, that are equal to 0. Then, variables in X are eliminated (using, e.g., Fourier-Motzkin elimination [Sch86]), which is denoted by the projection onto $X' \cup P$. The constraint is then intersected with the invariant of the destination location $I(q')$. Finally, clocks in X' are renamed with X, to get a constraint C' on X and P. A delay transition is obtained by delaying the constraint, using the time elapsing operation, and intersecting it with the invariant of the destination location $I(q)$.

In Figure 2.5, we present in a graphical way the computation of the successor constraint of a state (q,C). First, C is intersected with the guard g of the transition. Then, the clocks that must be reset by the transition (as in ρ) are projected onto zero. Then, the constraint is intersected with the invariant of the destination location $I(q')$. Time elapsing is then applied. The resulting constraint C' is finally obtained by intersecting again with the invariant of the destination location $I(q')$.

Definition 23. *A step of the semantics of a parametric timed automaton $\mathscr{A}(K)$ will be referred to as a* symbolic step *of $\mathscr{A}(K)$. Similarly, a run of the semantics of a parametric timed automaton $\mathscr{A}(K)$ will be referred to as a* symbolic run *of $\mathscr{A}(K)$.*

A symbolic run of $\mathscr{A}(K)$ is a finite alternating sequence of symbolic states and actions of the form $s_0 \overset{a_0}{\Rightarrow} s_1 \overset{a_1}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} s_m$, such that for all $i = 0,\ldots,m-1$,

and $a_i \in \Sigma$, $s_i \overset{a_i}{\Rightarrow} s_{i+1}$ is a symbolic step of $\mathscr{A}(K)$.

A symbolic run is represented under the form of a branch where states are depicted within nodes containing the name of the location and the constraint on the clocks and the parameters, and transitions are depicted using edges labeled with the name of the action.

Example 8. *Consider again the parametric timed automaton $\mathscr{A}$ of Example 5. Then Figure 2.6 depicts an example of a symbolic run of $\mathscr{A}$.*

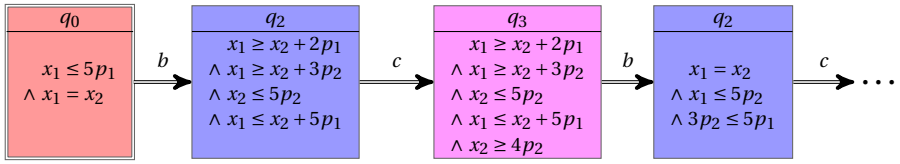


Figure 2.6: Example of a symbolic run for a parametric timed automaton

We give below some results on the constraints on the parameters associated with the symbolic runs of a parametric timed automaton, which will be used later on.

We first show that the constraint on the *parameters* associated with the symbolic states become more restrictive within a run, i.e., the constraint on the parameters of a given state of a run is included into the constraint on the parameters of a previous state of the same run. The parameter constraints associated to the reachable states can only get stronger, since the parameters do not evolve under the time elapse operation, and can only be further constrained by invariants or guard conditions.

Lemma 1. *Let $\mathscr{A}(K)$ be a parametric timed automaton, and R a symbolic run of $\mathscr{A}$ of the form $(q_0, C_0) \overset{a_0}{\Rightarrow} \cdots (q_i, C_i) \overset{a_i}{\Rightarrow} (q_{i+1}, C_{i+1}) \overset{a_{i+1}}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} (q_m, C_m)$. Then $C_{i+1}{\downarrow_P} \subseteq C_i{\downarrow_P}$, for all $0 \le i \le m-1$.*

Note that the above result does not mean that the constraints on the *clocks and the parameters* become more restrictive within a run, due to the elimination of clock variables. The relation $C_{i+1} \subseteq C_i$ does not hold in the general case.

We now state that, given a parametric timed automaton $\mathscr{A}(K)$, the constraint on the *parameters* associated with any symbolic state of $\mathscr{A}$ is included into K.

Lemma 2. *Let $\mathscr{A}(K)$ be a parametric timed automaton, and (q, C) a symbolic state of a symbolic run of A. Then $C{\downarrow_P} \subseteq K$.*

These two lemmas are the basis for the inverse method, described in Chapter 2.6.

Reachability and *Post* Computation

Recall from Definition 7 that a symbolic state s is reachable in one step from another symbolic state s' if s is the successor of s' in a symbolic run. This definition extends to sets of states. One defines $Post^i_{\mathscr{A}(K)}(S)$ as the set of states reachable from a set S of states in *exactly* i steps, and $Post^*_{\mathscr{A}(K)}(S)$ as the set of all states reachable from S in $\mathscr{A}(K)$ (i.e., $Post^*_{\mathscr{A}(K)}(S) = \bigcup_{i\geq 0} Post^i_{\mathscr{A}(K)}(S)$).

We are in particular interested in computing the set $Post^*_{\mathscr{A}(K)}(\{s_0\})$, where s_0 is the initial state of $\mathscr{A}(K)$. Note that if $Post^{i+1}_{\mathscr{A}(K)}(\{s_0\}) \sqsubseteq \bigcup_{j=0}^{i} Post^j_{\mathscr{A}(K)}(\{s_0\})$, then $Post^*_{\mathscr{A}(K)}(\{s_0\}) = \bigcup_{j=0}^{i} Post^j_{\mathscr{A}(K)}(\{s_0\})$.

Traces

The notion of trace associated with a concrete run, and the notion of trace set associated with a timed automaton apply in a straightforward manner to parametric timed automata.

Definition 24. *Given a parametric timed automaton $\mathscr{A}$ and a symbolic run r of $\mathscr{A}$ of the form $(q_0, C_0) \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} (q_m, C_m)$, the* trace associated with r *is the alternating sequence of locations and actions $q_0 \overset{a_0}{\Rightarrow} \cdots \overset{a_{m-1}}{\Rightarrow} q_m$. We say that location q_i, for $1 \leq i \leq m$, belongs to the trace.*

Note that the traces associated with symbolic runs of parametric timed automata are the same mathematical object (i.e., alternating sequences of locations and actions) as the traces associated with concrete runs of timed automata. As a consequence, we extend the notions of acyclicity and prefixes, defined for traces associated with concrete runs, to traces associated with symbolic runs. Moreover, we depict them under the same graphical form as traces associated with concrete runs, that is boxed nodes labeled with locations and double arrows labeled with actions.

Example 9. *The trace associated with the symbolic run of Example 8 is depicted in Figure 2.7.*

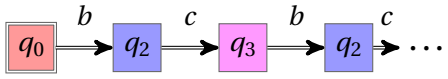


Figure 2.7: Example of a trace associated with a symbolic run of a parametric timed automaton

Definition 25. *We say that two (symbolic or concrete) runs are* equivalent, *if their associated trace is equal.*

Definition 26. *Given a parametric timed automaton $\mathscr{A}$, the* trace set of $\mathscr{A}$ *refers to the set of traces associated with the runs of $\mathscr{A}$.*

As for the traces associated with concrete runs, we extend the notion of acyclicity of a trace to trace sets, and say that a trace set is *acyclic* if all its traces are acyclic.

Example 10. *The trace set associated with the parametric timed automaton of Example 5 is depicted in Figure 2.8.*

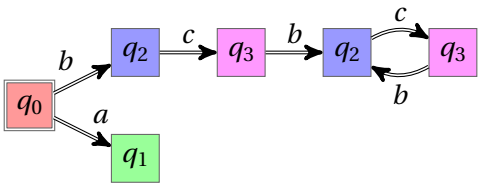


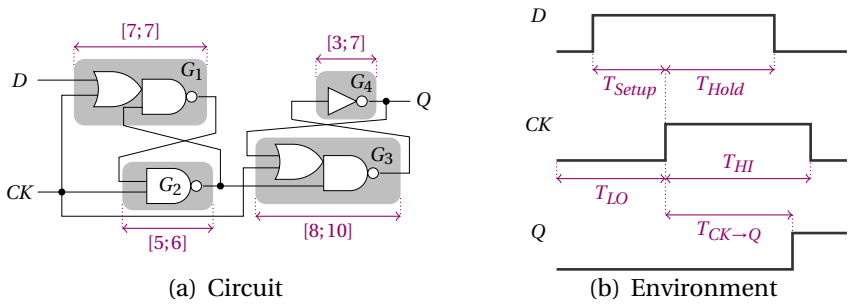
Figure 2.8: Example of a trace set of a parametric timed automaton

2.5 The Inverse Problem

2.5.1 A Motivating Example

Timed Automata is a powerful model and can be used for a wide range of case studies such as scheduling or the verification of hardware systems. As an example of the latter, consider the asynchronous "D flip-flop" circuit described in [CC07] and depicted in Figure 2.9(a). It is composed of 4 elements (G_1, G_2, G_3 and G_4) interconnected in a cyclic way. Elements G_1 and G_3 are made of an "OR" gate and a "NAND" gate. Element G_2 is a single "NAND" gate, and element G_4 is a single "NOT" gate (or inverter). The environment involves two input signals D and CK. The global output signal is Q. This system is a concurrent real-time system. It is concurrent, because each of the elements has its own behavior, that depends on the outputs of the other elements. It is real-time because each change of the output of an element occurs after some time, and events can be arbitrarily close to each other. The time between a change of the input and the change of the output is (usually) a matter of nanoseconds, but has a huge importance when verifying such systems.

We consider a bi-bounded inertial model for gates (see [BS95, MP95]), where any change of the input may lead to a change of the output (after some



(a) Circuit (b) Environment

Figure 2.9: Flip-flop circuit and its environment

delay). As a consequence, each gate G_i has a timing delay in the parametric interval $[\delta_i^-, \delta_i^+]$, with $\delta_i^- \leq \delta_i^+$. There are 4 other timing parameters (viz., $T_{HI}, T_{LO}, T_{Setup}$, and T_{Hold}) used to model the environment. The output signal of a gate G_i is denoted by g_i (note that $g_4 = Q$). The rising (resp. falling) edge of signal D is denoted by $D^{\nearrow}$ (resp. $D^{\searrow}$) and similarly for signals $CK, Q, g_1, \ldots, g_4$. We consider an environment starting from $D = CK = Q = 0$ and $g_1 = g_2 = g_3 = 1$, with the following ordered sequence of actions for inputs D and CK: $D^{\nearrow}$, $CK^{\nearrow}$, $D^{\searrow}$, $CK^{\searrow}$, as depicted in Figure 2.9(b). We consider that the behavior of this circuit is correct if, for this environment, the rise of signal Q (i.e., action $Q^{\nearrow}$) always occurs before the fall of signal CK (i.e., action $CK^{\searrow}$). The following question now arises: what are the possible values for these 12 timing parameters such that the circuit behaves in a correct way?

First, let us model this circuit using the formalism of parametric timed automata described in Chapter 2. Each gate is modeled by a parametric timed automaton, as well as the environment. Let us consider the example of element G_4 in Figure 2.9(a), which is a "NOT" gate (sometimes also referred to as an "inverter"). Recall that, in stable mode, the output (here, signal Q) of a "NOT" gate is equal to the inversion of the input (here, signal G_3). However, there may be configurations were both G_3 and Q are equal to each other, because of the delay between the change of the input and the change of the output. As a consequence, the parametric timed automaton modeling this "NOT" gate contains four locations $n_{00}, n_{01}, n_{10}, n_{11}$, where n_{ij} stands for a configuration of the gate where the input G_3 is equal to i and the output Q is equal to j. We give in Figure 2.10 the parametric timed automaton modeling this "NOT" gate. This parametric timed automaton contains one clock x and, as explained above, contains two parameters δ_3^- and δ_3^+, where $[\delta_3^-; \delta_3^+]$ represents the interval of time between a change of the input and the change of the output. Actions $G_3^{\nearrow}$ (resp. $G_3^{\searrow}$) denotes the rise (resp. the fall) of signal G_3, and similarly for Q. Other gates of the system can be modeled in a similar manner.

The reader interested in modeling hardware components with timed automata can refer to, e.g., [And10a, Chapter 2]. The parametric timed automaton $\mathscr{A}$ modeling the system results from the composition of those 5 parametric timed automata. Each location of $\mathscr{A}$ corresponds to a different value of the signals D, CK, g_1, g_2, g_3 and g_4 (recall that $g_4 = Q$).

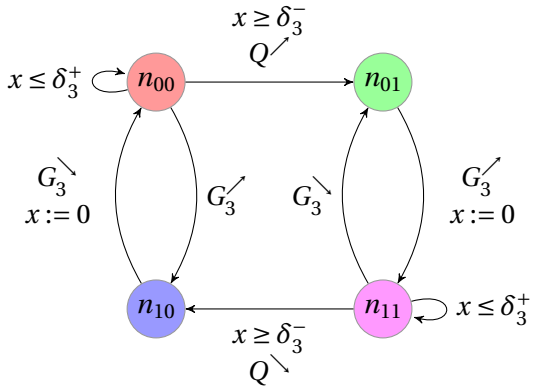


Figure 2.10: Parametric timed automaton modeling a "NOT" gate

The initial location q_0 corresponds to the initial levels of the signals according to the environment. Recall that we consider an environment starting from $D = CK = Q = 0$ and $g_1 = g_2 = g_3 = 1$, with the following ordered sequence of actions for inputs D and CK: $D^{\nearrow}$, $CK^{\nearrow}$, $D^{\searrow}$, $CK^{\searrow}$, as depicted in Figure 2.9(b) page 33. Therefore, we have the implicit constraint $T_{Setup} \leq T_{LO} \wedge T_{Hold} \leq T_{HI}$. The initial constraint K is:

$$T_{Setup} \leq T_{LO} \wedge T_{Hold} \leq T_{HI} \wedge \bigwedge_{i=1,..,4} \delta_i^- \leq \delta_i^+$$

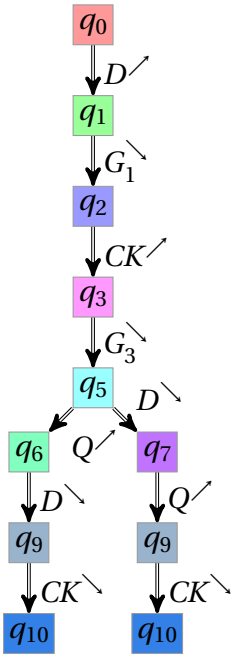
We consider that the circuit has a *good behavior* if it verifies the following property $Prop_1$: "every trace contains both $Q^{\nearrow}$ and $CK^{\searrow}$, and $Q^{\nearrow}$ occurs before $CK^{\searrow}$". That is, the raise of signal Q must occur before the fall of signal CK. We consider the following valuation π_0 of the parameters[1]:

$$\begin{array}{cccc} T_{HI} = 24 & T_{LO} = 15 & T_{Setup} = 10 & T_{Hold} = 17 \\ \delta_1^- = 7 & \delta_1^+ = 7 & \delta_2^- = 5 & \delta_2^+ = 6 \\ \delta_3^- = 8 & \delta_3^+ = 10 & \delta_4^- = 3 & \delta_4^+ = 7 \end{array}$$

Let us study the behavior of the flip-flop circuit under π_0. The trace set of $\mathscr{A}[\pi_0]$ is depicted in Figure 2.11(a), where the meaning of each location in terms of signals is given in Figure 2.11(b). Recall that we do not depict each trace separately, but depict the trace set under the form of a tree or a graph.

[1] We are not specifically interested in the units here, but it could be nanoseconds.

However, this graph structure is only used for the sake of simplicity of representation of the possible traces, and does not contain any information on the possible branching behavior of the system.



(a) Trace set

Location	D	CK	g_1	g_2	g_3	g_4
q_0	0	0	1	1	1	0
q_1	1	0	1	1	1	0
q_2	1	0	0	1	1	0
q_3	1	1	0	1	1	0
q_4	0	1	0	1	1	0
q_5	1	1	0	1	0	0
q_6	1	1	0	1	0	1
q_7	0	1	0	1	0	0
q_8	0	0	0	1	0	0
q_9	0	1	0	1	0	1
q_{10}	0	0	0	1	0	1

(b) Locations

Figure 2.11: Trace set of the flip-flop circuit under π_0

Each of the two traces depicted in this trace set contains both $Q^{\nearrow}$ and $CK^{\searrow}$, and $Q^{\nearrow}$ occurs before $CK^{\searrow}$. As a consequence, this trace set is a *good* trace set according to the property we want to verify. We are now interested in studying the evolution of the behavior of the system if one changes some of the values of the parameters. More precisely, we are interested in identifying parameter valuations for which the system has exactly the same (good) behavior, i.e., exactly the same trace set.

2.5.2 The Problem

The *inverse problem* can be stated as follows [ACEF09]:

The Inverse Problem
Given a parametric timed automaton $\mathscr{A}$ and a reference valuation π_0, find a constraint K_0 on the parameters such that:

- $\pi_0 \models K_0$, and
- for all $\pi \models K_0$, the trace sets of $\mathscr{A}[\pi]$ and $\mathscr{A}[\pi_0]$ are the same.

This problem considers an equality of trace sets between $\mathscr{A}[\pi]$ and $\mathscr{A}[\pi_0]$, and thus guarantees a *time-abstract equivalence* between the behavior of $\mathscr{A}[\pi]$ and $\mathscr{A}[\pi_0]$.

2.6 The Inverse Method Algorithm

2.6.1 Principle

We recall in the following the *inverse method* [ACEF09], which is a solution to the inverse problem stated above. The inverse method consists in generating runs starting from the initial state, and removing states incompatible with the reference values by appropriately refining the current constraint K_0 on the parameters. The generation procedure is then restarted until a new incompatible state is produced, and so on, iteratively until no incompatible state is generated.

The inverse method algorithm $IM(\mathscr{A}, \pi_0)$ is given in Algorithm 1. Starting with $K = \texttt{true}$, it iteratively computes a growing set of reachable states. It is made of two **do** loops. The inner **do** loop (lines 3–7) removes all the π_0-incompatible states for a given depth i of the runs, i.e., states of $Post^i_{\mathscr{A}(K)}(\{s_0\})$. This removal is performed by removing a π_0-incompatible inequality randomly selected within the projection onto the parameters of the constraint C associated with a π_0-incompatible state; recall that this projection onto the parameters, i.e., elimination of the clocks, is denoted by $C{\downarrow_P}$ in the algorithm (line 5). The outer **do** loop (lines 2–11) iteratively computes the set of all reachable states. When the fixpoint is reached, i.e., when all new states computed at iteration i are *equal* to states computed at previous iterations ($Post_{\mathscr{A}(K)}(S) \sqsubseteq S$), the intersection K_0 of all the constraints on the parameters associated with the states of S is returned.

The two major steps of the algorithm are the following ones:

1. the iterative negation of the π-incompatible states (by negating a π-incompatible inequality J) prevents for any $\pi \models K_0$ the behavior different from π;

Algorithm 1: Inverse method algorithm $IM(\mathscr{A}, \pi_0)$

input : Parametric timed automaton $\mathscr{A}$ of initial state s_0
input : Valuation π_0 of the parameters
output: Constraint K_0 on the parameters

$i \leftarrow 0$; $K \leftarrow \texttt{true}$; $S \leftarrow \{s_0\}$
while `true` **do**
  while *there are π_0-incompatible states in S* **do**
    Select a π_0-incompatible state (q, C) of S (i.e., s.t. $\pi_0 \not\models C$) ;
    Select a π_0-incompatible inequality J in $C{\downarrow_P}$ (i.e., s.t. $\pi_0 \not\models J$) ;
    $K \leftarrow K \wedge \neg J$;
    $S \leftarrow \bigcup_{j=0}^{i} Post^j_{\mathscr{A}(K)}(\{s_0\})$;
  if $Post_{\mathscr{A}(K)}(S) \sqsubseteq S$ **then**
    return $K_0 \leftarrow \bigcap_{(q,C)\in S} C{\downarrow_P}$
  $i \leftarrow i + 1$;
  $S \leftarrow S \cup Post_{\mathscr{A}(K)}(S)$

2. the intersection of the constraints associated with all the reachable states guarantees that all the behaviors under π are allowed for all $\pi \models K_0$.

2.6.2 A Toy Example

Let us consider the parametric timed automaton given in Figure 2.12, following the formalism defined in Chapter 2.4. Let us assume that q_2 corresponds to a "bad location". Using this information, classical methods, such as CEGAR methods [CGJ+00], will synthesize the constraint $Z : p_1 < p_3$, which guarantees that the location is not reachable. Suppose now that we are given the following "good" parameter valuation $\pi_0 : \{p_1 = 4, p_2 = 2, p_3 = 6\}$, under which the parametric timed automaton is assumed to have a "good" behavior. Then the inverse method will synthesize the constraint $K_0 : p_1 < p_3 \wedge p_2 \leq p_1$. For all valuations π of the parameters satisfying K_0, the inverse method guarantees that the parametric timed automaton behaves in the same manner as under π_0. We are thus ensured that the behavior of the parametric timed automaton is correct. We can note that K_0 is strictly smaller than Z. On the one hand, this may be viewed as a limitation of the inverse method. On the other hand, this may indicate that there are incorrect behaviors other than those corresponding to the reaching of q_2. For example, there are some parameter valuations satisfying Z, under which a blocking situation (deadlock) of the parametric timed automaton occurs at the initial location q_0. In contrast, the inverse method guarantees

that such a situation is impossible under any instance satisfying K_0 (because this situation does not occur under π_0).

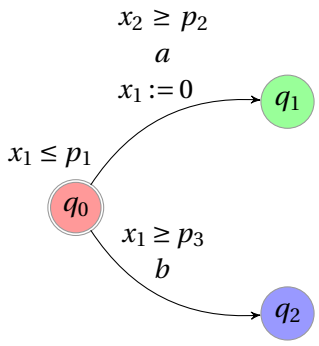


Figure 2.12: A toy parametric timed automaton

2.7 Behavioral Cartography of Timed Automata

Outline of the Chapter In Section 2.7.1, we introduce the behavioral cartography algorithm. We present a case study in Section 2.7.3, analyzed using IMITATOR.

2.7.1 The Behavioral Cartography Algorithm

By iterating the inverse method *IM* of Section 2.6 over all the *integer* points of a rectangle[2] V_0 (of which there are a finite number), one is able to decompose (most of) the parametric space included into V_0 into behavioral tiles. We give the behavioral cartography algorithm *BC* in Algorithm 2 [AF10].

Note that two tiles with distinct trace sets are necessarily disjoint. On the other hand, two tiles with the same trace sets may overlap. In many cases, all the real-valued space of V_0 is covered by *Tiling* (see the flip flop case study in Section 2.7.3). Moreover, the space covered by *Tiling* often largely exceeds the limits of V_0.

If now a decidable trace property is given then one can check which tiles are good (i.e., the tiles whose trace set satisfies the property), and which ones are bad. One can thus partition the rectangle V_0 into a good (resp. bad) subspace, i.e., a union of good (resp. bad) tiles.

A major advantage of the behavioral cartography is that the cartography itself does not depend on the property one wants to check. Only the partition

[2] Actually, V_0 can be more generally a convex set containing a finite number of integer points.

Algorithm 2: Behavioral cartography algorithm $BC(\mathscr{A}, V_0)$

input : A parametric timed automaton $\mathscr{A}$
input : A finite rectangle $V_0 \subseteq \mathbb{R}_+^M$
output: *Tiling*: list of tiles (initially empty)

```
1 repeat
2     select an integer point π ∈ V_0;
3     if π does not belong to any tile of Tiling then
4         Add IM(A, π) to Tiling;
5 until Tiling contains all the integer points of V_0;
```

between good and bad tiles involves the considered property. Moreover, the algorithm is interesting because one does not need to compute the set of all the reachable states. On the contrary, each call to the inverse method algorithm quickly reduces the state space by removing the incompatible states. This allows us to overcome the state space explosion problem, which prevents other methods to terminate in practice. Also note that the cartography algorithm makes use of no approximation.

General Case

For the general case (i.e., possibly cyclic parametric timed automata), it is also possible to identify classes of systems for which the full coverage of the rectangle V_0 is guaranteed using the classical version *BC* of the behavioral cartography algorithm.

2.7.2 Finite Cartography

In this section, we give sufficient conditions on the finiteness of the number of tiles in the cartography. We suppose here that the coefficients in the guards and invariants can only be in $\mathbb{Z}$, the set of all integers.

Proposition 1. *Constraints generated by the Inverse Method can only be conjunctions of linear constraints on the parameters.*

Proof. The proof is straightforward by considering that for a state $q = (l, C)$, C is a conjuction of guards and invariants. In our framework, guards and invariants can only be linear constraints. Therefore, for every state $q = (l, C)$, $\exists X$:C is a conjuction of guards and invariants and $K = \bigcap_{(q,C)\in S} \exists X$:$C$ also is a conjuction of guards and invariants □

Under our assumption, linear constraint are of the form $\sum_{i\in 1..n}(\alpha_i * p_i)$, with $\alpha_i \in \mathbb{Z}$ for all $i \in I$. Let $(\alpha_1,...\alpha_n) \in \mathscr{P}^n \subset \mathbb{Z}$, $c \in \mathbb{Z}$ and $\prec \in \mathscr{O} = \{<, \leq, >, \geq\}$.
Let C be the constraint $\sum_{i\in 1..n}(\alpha_i * p_i) \prec c$.
We denote by $\mathscr{C}_{(\alpha_1,...\alpha_n,c,\prec)} = \{(v_1,...,v_n) \in \mathbb{R}^n | \sum_{i\in 1..n}(\alpha_i * v_i) \prec c\}$ the set of all points of $\mathbb{R}^n$ that satisfy C.

Proposition 2. *Let $V_0 = I_1 \times ... \times I_n$ with I_i a bounded interval of $\mathbb{R}$ for all $i \in \{1..n\}$.*
Let $E = \{(\alpha_1,...\alpha_n, c, \prec) \in \mathscr{P}^n \times \mathbb{Z} \times \mathscr{O} | \mathscr{C}_{(\alpha_1,...\alpha_n,c,\prec)} \cap V_0 \neq \emptyset$ et $\mathscr{C}_{(\alpha_1,...\alpha_n,c,\prec)} \cap V_0 \neq V_0\}$

If $\mathscr{P}$ is bounded then $|E| \leq +\infty$

E corresponds to the set of all non trivial constraints of V_0.

Proof. To show that $|E| < +\infty$, we need to prove that if $(\alpha_1,...\alpha_n, c, \prec) \in E$ then c belongs to a bounded interval of $\mathbb{Z}$. Indeed, $\mathscr{P}$ is bounded and $|\mathscr{O}| = 4$, then if c can only take a finite number of values then E is finite as well.
Let $(\alpha_1,...\alpha_n, c, \prec) \in E$.
$\exists (v_1,...,v_n) \in V_0 | \sum_{i\in 1..n}(\alpha_i * v_i) \prec c$ because $\mathscr{C}_{(\alpha_1,...\alpha_n,c,\prec)} \cap V_0 \neq \emptyset$
and
$\exists (v'_1,...,v'_n) \in V_0 | \sum_{i\in 1..n}(\alpha_i * v'_i)(\neg \prec) c$ because $\mathscr{C}_{(\alpha_1,...\alpha_n,c,\prec)} \cap V_0 \neq V_0$
Without loss of generality, we have that,
$c \leq \sum_{i\in 1..n}(\alpha_i * v_i)$ and $c \geq \sum_{i\in 1..n}(\alpha_i * v'_i)$.
V_0 is bounded as a finite cartesian product of bounded parts, therefore there exists $M \in \mathbb{N}$ such that $V_0 \subseteq [-M; M]^n$.
But we have that

$$c \leq \sum_{i\in 1..n}(\alpha_i * v_i)$$

which can be rewritten as

$$c \leq \sum_{i\in 1..n | \alpha_i > 0}(\alpha_i * v_i) + \sum_{i\in 1..n | \alpha_i < 0}(\alpha_i * v_i)$$

$$c \leq \sum_{i\in 1..n | \alpha_i > 0}(\alpha_i * v_i)$$

Then,

$$c \leq \sum_{i\in 1..n | \alpha_i > 0}(\alpha_i * M)$$

We also have that $\mathscr{P}$ is bounded. Therefore, there exists $M' in \mathbb{N}$ suche that $\mathscr{P} \subseteq [-M';M']$.
Therefore,

$$c \leq \sum_{i \in 1..n | \alpha_i > 0} (M' * M)$$

We have that $|\{i \in 1..n | \alpha_i > 0\}| \leq n$, therefore

$$c \leq n * M' * M$$

Moreover, we have that

$$c \geq -n * M' * M \square$$

□

We know that a tile E is an area delimited by a finite number of constraints on the parameters. But, there only exists a finite number of constraints that do not define a non trivial tile of V_0. Therefore, we have proven the following theorem.

Theorem 1. *If the coefficients on the constraints that can be generated by the Inverse Method on the PTA $\mathscr{A}$ are bounded, then the Behavioral Cartography has a finite number of tiles.*

Proof. The proof is straightforward from Property 2. □

2.7.3 Case Study: Flip-flop

We consider a of case study and synthesize constraints. The constraint synthesized by the behavioral cartography are then compared with constraints from the literature, when applicable. For a fully detailed description and more case studies, refer to [And10b].

We apply here the behavioral cartography to the flip-flop example described in Section 2.5.1. For the sake of simplicity, we consider a model with only 2 parameters, with the following V_0:

$$\delta_3^+ \in [8,30] \quad \text{and} \quad \delta_4^+ \in [3,30].$$

The other parameters are instantiated as follows:

$$T_{HI} = 24 \quad T_{LO} = 15 \quad T_{Setup} = 10 \quad T_{Hold} = 17 \quad \delta_1^- = 7$$
$$\delta_1^+ = 7 \quad \delta_2^- = 5 \quad \delta_2^+ = 6 \quad \delta_3^- = 8 \quad \delta_4^- = 3$$

We compute the cartography of the flip-flop circuit according to δ_3^+ and δ_4^+, depicted in Figure 2.13. The dashed rectangle corresponds to V_0.

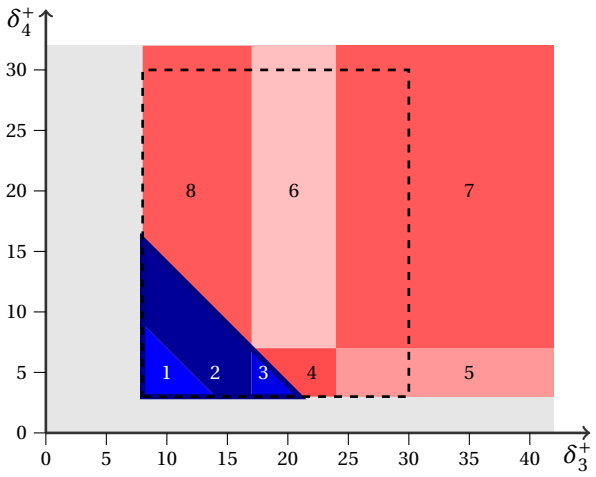


Figure 2.13: Behavioral cartography of the flip-flop according to δ_3^+ and δ_4^+

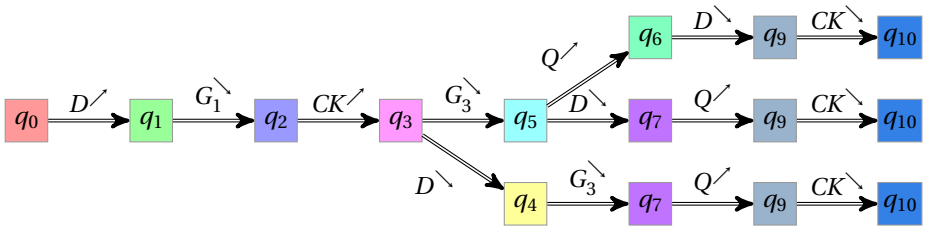


Figure 2.14: Trace set of tile 3 for the flip-flop case study

First note that the whole (real-valued) V_0 is covered. Note also that tiles 5 to 8 are unbounded. Actually, this cartography covers the whole[3] real-valued parametric space $\mathbb{R}_+ \times \mathbb{R}_+$. According to the nature of the trace sets, we can easily partition the tiles into good and bad tiles with respect to property $Prop_1$. For example, the trace set of tile 3 (corresponding to the constraint $\delta_3^+ + \delta_4^+ < 24 \wedge \delta_3^+ \geq 17 \wedge \delta_4^+ \geq 3$) is given in Figure 2.14, where the meaning of each location in terms of signals is given in Figure 2.11(b) page 35. This tile is a *good* tile because $Q^{\nearrow}$ occurs before $CK^{\searrow}$ for all traces.

The trace set of tile 7 (corresponding to the constraint $\delta_3^+ \geq 24 \wedge \delta_4^+ \geq 7$) is given in Figure 2.15. This is a *bad* tile because there exist traces where $Q^{\nearrow}$ occurs after $CK^{\searrow}$.

One sees more generally that tiles 1 to 3 are good while tiles 4 to 8 are bad. From this partition into good and bad tiles, we infer the following constraint:

$$\delta_3^+ + \delta_4^+ \leq 24 \ \wedge \ \delta_3^+ \geq 8 \ \wedge \ \delta_4^+ \geq 3$$

[3] Apart from the irrelevant zone originating from the model ($\delta_3^+ < 8$ or $\delta_4^+ < 3$).

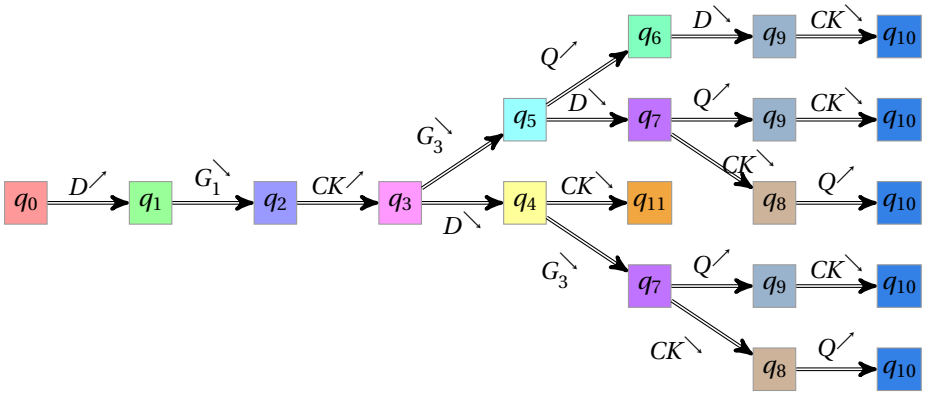


Figure 2.15: Trace set of tile 7 for the flip-flop case study

which gives the *maximal* set of good parameters, thus solving the good parameters problem for this example.

Comparison with other methods. By computing in a brute manner the whole set of reachable states for all possible valuations of the parameters, and performing the intersection with the set of bad locations, we get the same constraint ensuring the good behavior of the system. Note that this comparison is possible because this example is rather simple; for bigger examples, such a computation would be impossible because of the state space explosion problem (see the cartography Root Contention Protocol in [AS13]). In [CC07], a constraint Z guaranteeing a good behavior is given (see Section 2.7.3). The projection of this constraint Z onto δ_3^+ and δ_4^+ gives

$$\delta_3^+ < 11 \wedge \delta_3^+ + \delta_4^+ < 18 \wedge \delta_3^+ \geq 8 \wedge \delta_4^+ \geq 3,$$

which is strictly included in the constraint synthesized by *IM*[4].

[4]Actually, the comparison is not completely fair, because the two models are slightly different: in particular, the authors of [CC07] consider an environment where D is initially equal to either 0 or to 1.

Chapter 3

State Merging in Parametric Timed Automata

A fundamental problem in the exploration of the reachability space in PTAs is to compact as much as possible the generated space of symbolic states. We introduce in this chapter a state merging technique based on convex union. Roughly speaking, two states are merged when their discrete part is the same, and the union of their respective continuous part (values of the clocks and parameters) is convex. On the one hand, this technique often considerably reduces the state space. On the other hand, exploring the state space using this technique does not reflect the standard semantics of PTAs: the set of possible paths is an over-approximation of the set of paths in the original PTAs semantics. However, we show that the state space computed using the merging reduction preserves the set of reachable locations and executable actions. That is, the sets of reachable locations and executable actions obtained using the merging reduction are the same as those obtained using the classical semantics.

Moreover, we show that *IM* equipped with our merging reduction (called IM_{Mrg}) does not preserve traces anymore; however, it preserves locations (i.e., discrete reachability), and outputs a weaker constraint. However, we show that actions are not preserved in the general case. We exhibit a subclass of PTAs, namely backward-deterministic PTA, for which action preservation is guaranteed. Furthermore, we show that IM_{Mrg} outputs a weaker constraint (i.e., a larger set of parameter valuations) than *IM*, which is interesting.

Finally, we define a new version IM'_{Mrg} of IM_{Mrg} that preserves not only locations but actions too, at the cost of a more restrictive constraint than IM_{Mrg}, but still weaker than *IM*. Our work is implemented in IMITATOR [AFKS12] and shows large state space reductions in many cases, especially for scheduling problems. Finally, and more surprisingly, the time overhead induced by the convexity test is often not significant in the few case studies where the state

space is not reduced.

Outline We define and characterize the merging reduction in Section 3.2. Section 3.3 is dedicated to *IM* combined with the merging reduction. We give experiments in Section 3.4 of the merging reduction that will be presented in Section 3.3.

3.1 General Results for Parametric Timed Automata

We prove below several general results on constraints and PTAs, that will be used in the forthcoming proofs.

We first prove below a simple result on variable elimination.

Lemma 3 (Distributivity of $\downarrow$ over $\cup$)**.** *Let $C_1, C_2 \in \mathscr{L}(X \cup P)$. Then:*

$$(C_1 \cup C_2)\downarrow_P = C_1\downarrow_P \cup C_2\downarrow_P$$

Proof. $(C_1 \cup C_2)\downarrow_P = \{\pi \mid \pi \models C_1 \cup C_2\}$
$= \{\pi \mid \pi \models C_1 \vee \pi \models C_2\}$
$= \{\pi \mid \pi \models C_1\} \cup \{\pi \mid \pi \models C_2\}$
$= C_1\downarrow_P \cup C_2\downarrow_P$ □

Lemma 4 (Preservation of inclusion)**.** *Let $\mathscr{A}$ be a PTA, and $(l_1, C_1), (l_2, C_2) \in Reach^*(\mathscr{A})$ with $(l_1, C_1) \overset{a}{\Rightarrow} (l_2, C_2)$. Let C_1' be such that $C_1 \subseteq C_1'$.*

Then $\exists C_2'$ such that $C_2 \subseteq C_2'$ and $(l_1, C_1') \overset{a}{\Rightarrow} (l_2, C_2')$.

Proof. From the semantics of PTAs, we have that

$$C_2 = \Big(\big(\big(C_1(X) \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\big)_{[X' \leftarrow X]}\Big)\uparrow \wedge I(l')(X)$$

and

$$C_2' = \Big(\big(\big(C_1'(X) \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\big)_{[X' \leftarrow X]}\Big)\uparrow \wedge I(l')(X)$$

We have $C_1 \subseteq C_1'$; then C_1' (resp. C_2') is obtained from C_1 (resp. C_2) by adding other constraints, which are the same in each expression, hence this preserves inclusion. Then a projection is performed onto $X' \cup P$, which preserves inclusion. Then the expressions are intersected with another constraint ($I(l')(X')$), which preserves the inclusion too. Then the expressions have their (same) set of clocks renamed, and are intersected with the same destination constraint $I(l')(X)$, which again preserves inclusion. Hence $C_2 \subseteq C_2'$. □

Lemma 5 (Disjunction and Post)**.** *Let $\mathscr{A}$ be a PTA, and $(l,C_1),(l,C_2) \in Reach^*(\mathscr{A})$. Suppose $(l,C_1) \overset{a}{\Rightarrow} (l',C_1')$, $(l,C_2) \overset{a}{\Rightarrow} (l',C_2')$ and $(l,C) \overset{a}{\Rightarrow} (l',C')$. Then:*

$$C = C_1 \cup C_2 \Longrightarrow C' = C_1' \cup C_2'.$$

Proof. The proof follows from the semantics of PTAs.

$C = C_1 \cup C_2$

$\Longrightarrow \big(C \wedge g(X) \wedge X' = \rho(X)\big) = \Big(\big(C_1 \wedge g(X) \wedge X' = \rho(X)\big) \cup \big(C_2 \wedge g(X) \wedge X' = \rho(X)\big)\Big)$
(distributivity of $\wedge$ over $\cup$)

$\Longrightarrow \big(C \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} = \Big(\big(C_1 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \cup \big(C_2 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P}\Big)$ (Lemma 3)

$\Longrightarrow \Big(\big(C \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big) = \Big(\Big(\big(C_1 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big) \cup \Big(\big(C_2 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)\Big)$ (distributivity of $\wedge$ over $\cup$)

$\Longrightarrow \Big(\big(C \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]} = \Big(\Big(\big(C_1 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]} \cup \Big(\big(C_2 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]}\Big)$
(variables renaming)

$\Longrightarrow \Big(\Big(\big(C \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]}\Big)\uparrow = \Big(\Big(\Big(\big(C_1 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]}\Big)\uparrow \cup \Big(\Big(\big(C_2 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]}\Big)\uparrow\Big)$ (Lemma 3 + distributivity of $\wedge$ over $\cup$)

$\Longrightarrow \Big(\Big(\big(C \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]}\Big)\uparrow = \Big(\Big(\Big(\big(C_1 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]}\Big)\uparrow \cup \Big(\Big(\big(C_2 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]}\Big)\uparrow\Big)$ (Lemma 3 + distributivity of $\wedge$ over $\cup$)

$\Longrightarrow \Big(\Big(\big(C \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]}\Big)\uparrow \wedge I(l')(X) = \Big(\Big(\Big(\big(C_1 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]}\Big)\uparrow \wedge I(l')(X) \cup \Big(\Big(\big(C_2 \wedge g(X) \wedge X' = \rho(X)\big)\downarrow_{X' \cup P} \wedge I(l')(X')\Big)_{[X' \leftarrow X]}\Big)\uparrow \wedge I(l')(X)\Big)$ (distributivity of $\wedge$ over $\cup$)

$\Longrightarrow C' = C_1' \cup C_2'.$

□

3.2 Merging States in Parametric Timed Automata

3.2.1 Principle

The principle of our method relies on the fact that only convex polyhedra are tractable. We will merge two states if their discrete parts are identical (meaning they model the same state of the system) and if the union of their corresponding constraint is convex. In short, two states are mergeable if they share the same discrete part, and the union of their corresponding constraints is convex.

Definition 27 (Merging)**.** *Two states* $s_1 = (l_1, C_1)$ *and* (l_2, C_2) *are* mergeable *if* $l_1 = l_2$ *and* $C_1 \cup C_2$ *is convex; then,* $(l_1, C_1 \cup C_2)$ *is their* merging *denoted by* $merge(s_1, s_2)$.

Given a set S of states, $Merge(S)$ denotes the result of applying iteratively the merging of a pair of states of S until no further merging applies, as given in Algorithm 3.

Algorithm 3: Merging a set of states

input : Set S of states
output: Merged set S of states

1 $Q \leftarrow S$;
2 **while** $\exists (l, C_1), (l, C_2) \in Q$ *such that* $C_1 \neq C_2$ *and* $C_1 \cup C_2$ *is convex* **do**
3 Choose $(l, C_1) \in Q$, $(l, C_2) \in Q$ such that $C_1 \neq C_2$ and $C_1 \cup C_2$ is convex ;
4 $Q \leftarrow Q \setminus \{(l, C_1), (l, C_2)\} \cup \{merge((l, C_1), (l, C_2))\}$;
5 **return** Q

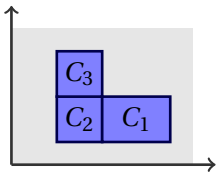


Figure 3.1: Non-determinism of merging

Remark. This process is not deterministic , i.e., the result depends on the order of the iterative merging operations of pairs of states. Consider three states $(l, C_1), (l, C_2), (l, C_3)$ such that $C_1 \cup C_2$ and $C_2 \cup C_3$ are convex, but $C_1 \cup C_3$ is not. This situation is depicted in Fig. 3.1 with 2 parameter dimensions. In that case, two possible sets of states can result from an application of the merging to these 3 states. That is, either $\{(l, C_1), (l, C_2 \cup C_3)\}$ or $\{(l, C_1 \cup C_2), (l, C_3)\}$.

3.2.2 Merging and Reachability

We define below the semantics of PTAs with merging.

Definition 28 (Semantics of PTAs with merging)**.** *Let* $\mathscr{A} = (\Sigma, L, l_0, X, P, K, I, \rightarrow)$ *be a PTA. The* semantics of $\mathscr{A}$ with merging *is* $\mathrm{L}_{Mrg}(\mathscr{A}) = (\Sigma, S, S_0, \Rightarrow_{Mrg})$ *where*

$$S = \{(l, C) \in L \times \mathscr{L}(X \cup P) \mid C \subseteq I(l)\},$$
$$S_0 = \{(l_0, K \wedge I(l_0) \wedge \bigwedge_{i=1}^{H-1} x_i = x_{i+1})\}$$

and a transition $(l, C) \overset{a}{\Rightarrow} (l', C')$ *belongs to* $\Rightarrow_{Mrg}$ *if there exists* $n \in \mathbb{N}$ *such that*

- $(l, C) \in ReachM^n$*, and*
- $(l', C') \in ReachM^{n+1}$*,*

where $ReachM^n$ *is inductively defined as follows:*

- $ReachM^0 = S_0$*, and*
- $ReachM^{i+1} = Merge\big(Post_{\mathscr{A}}(ReachM^i)\big)$ *for all* $i \in \mathbb{N}$*.*

Recall that *Post* is defined using the $\Rightarrow$ relation of $\mathscr{A}$ without merging. Hence the semantics of PTAs with merging iteratively computes states (using the standard transition relation), and merges the new states at each iteration.

Then we define $\Rightarrow^i_{Mrg}$, *PostM*, $ReachM^*$, *PathsM*, *TracesM*, *LocationsM* and *ActionsM* the same way as $\Rightarrow^i$, *Post*, $Reach^*$, *Paths*, *Traces*, *Locations* and *Actions*, respectively, by replacing within their respective definition $\Rightarrow$ with $\Rightarrow_{Mrg}$. Observe that, from the definition of $\Rightarrow_{Mrg}$ in Definition 28, *PostM* can be defined as *Post* followed by *Merge*, i.e., $PostM = Merge \circ Post$.

3.2.3 Characterization of the Merging Reduction

The following lemma states that the initial state of any path (hence, including of length 0) of $\mathscr{A}$ without merging is the same for $\mathscr{A}$ with merging.

Lemma 6. *Let* $\mathscr{A}$ *be a PTA. Then* $Reach^0(\mathscr{A}) = ReachM^0(\mathscr{A})$*.*

Proof. From Definitions 22 and 28. □

The main property preserved by merging states while generating the reachability graph is the preservation of each time-abstract transition, i.e., taken one by one. In other words, for each time-abstract transition $l_1 \overset{a}{\Rightarrow} l_2$ in the graph obtained without merging, there is a corresponding time-abstract transition

$l_1 \overset{a}{\Rightarrow} l_2$ in the graph obtained with merging. However, this does not extend to traces.

The characterization of merging will be stated in Theorem 2. This result relies on the two forthcoming lemmas 8 and 10.

First, the following lemma will be used to prove Lemma 8 by induction.

Lemma 7. *Let $\mathscr{A}$ be a PTA. Let $(l_0, C_0) \overset{a_0}{\Rightarrow} \cdots \overset{a_{n-1}}{\Rightarrow} (l_n, C_n) \in Paths(\mathscr{A})$, and $(l_0, C'_0) \overset{a_0}{\Rightarrow}_{Mrg} \cdots \overset{a_{n-1}}{\Rightarrow}_{Mrg} (l_n, C'_n) \in PathsM(\mathscr{A})$ such that $C_i \subseteq C'_i$, for all $0 \leq i \leq n$.*

Suppose $(l_n, C_n) \overset{a_n}{\Rightarrow} (l_{n+1}, C_{n+1}) \in \Rightarrow(\mathscr{A})$. Then there exists C'_{n+1} such that

1. *$(l_n, C'_n) \overset{a_n}{\Rightarrow}_{Mrg} (l_{n+1}, C'_{n+1}) \in \Rightarrow_{Mrg}(\mathscr{A})$, and*
2. *$C_{n+1} \subseteq C'_{n+1}$.*

Proof. Since $C_n \subseteq C'_n$ and $(l_n, C_n) \overset{a_n}{\Rightarrow} (l_{n+1}, C_{n+1}) \in \Rightarrow(\mathscr{A})$, then there exists C''_{n+1} such that $(l_n, C'_n) \overset{a_n}{\Rightarrow} (l_{n+1}, C''_{n+1})$ and $C_{n+1} \subseteq C''_{n+1}$ (from Lemma 4). We consider two cases, depending whether (l_{n+1}, C''_{n+1}) is merged or not in *PostM*.

- Suppose (l_{n+1}, C''_{n+1}) is not merged with another state. That is $(l_{n+1}, C''_{n+1}) \in Post_{\mathscr{A}}(ReachM^n(\mathscr{A}))$ and since it is not merged, then (l_{n+1}, C''_{n+1}) belongs to $PostM_{\mathscr{A}}(ReachM^n(\mathscr{A})) = ReachM^{n+1}(\mathscr{A})$. Hence the result holds, with $C'_{n+1} = C''_{n+1}$.
- Suppose (l_{n+1}, C''_{n+1}) is merged with other states, thus creating new state (l_{n+1}, C'_{n+1}), with $C'_{n+1} = C''_{n+1} \cup C'''_{n+1}$ Then $(l_n, C'_n) \overset{a_n}{\Rightarrow}_{Mrg} (l_{n+1}, C'_{n+1}) \in \Rightarrow_{Mrg}(\mathscr{A})$, and $C_{n+1} \subseteq C''_{n+1} \subseteq C'_{n+1}$.

□

Now, we can prove that a trace in a non-merged reachability graph has an equivalent trace in a merged reachability graph.

Lemma 8 (Merging and reachability ($\Longrightarrow$))**.** *Let $\mathscr{A}$ be a PTA. Let $(l_0, C_0) \overset{a_0}{\Rightarrow} \cdots \overset{a_{n-1}}{\Rightarrow} (l_n, C_n) \in Paths(\mathscr{A})$. Then there exist $C'_1, \ldots, C'_n$ such that:*

1. *$(l_0, C_0) \overset{a_0}{\Rightarrow}_{Mrg} (l_0, C'_1) \overset{a_1}{\Rightarrow}_{Mrg} \cdots \overset{a_{n-1}}{\Rightarrow}_{Mrg} (l_n, C'_n) \in PathsM(\mathscr{A})$, and*
2. *$C_i \subseteq C'_i$, for all $1 \leq i \leq n$.*

Proof. By induction on n.

- **Base case.** For $n = 0$, we have a path reduced to a single state (l_0, C_0). By lemma 6, $(l_0, C_0) \in PathsM(\mathscr{A})$.

- **Induction step.** Suppose that $n \geq 0$ and the property holds for n. Let $(l_0, C_0) \overset{a_0}{\Rightarrow} \cdots \overset{a_n}{\Rightarrow} (l_{n+1}, C_{n+1}) \in \mathit{Paths}(\mathscr{A})$. By induction hypothesis, there exist $C'_1, \ldots, C'_n$ such that $(l_0, C_0) \overset{a_0}{\Rightarrow}_{Mrg} (l_0, C'_1) \overset{a_1}{\Rightarrow}_{Mrg} \cdots \overset{a_{n-1}}{\Rightarrow}_{Mrg} (l_n, C'_n) \in \mathit{PathsM}(\mathscr{A})$, and $C_i \subseteq C'_i$, for all $0 \leq i \leq n$.

  From Lemma 7, there exists C'_{n+1} such that

  1. $(l_n, C'_n) \overset{a_n}{\Rightarrow}_{Mrg} (l_{n+1}, C'_{n+1}) \in \Rightarrow_{Mrg}(\mathscr{A})$, and
  2. $C_{n+1} \subseteq C'_{n+1}$.

  Hence $(l_0, C_0) \overset{a_0}{\Rightarrow}_{Mrg} (l_0, C'_1) \overset{a_1}{\Rightarrow}_{Mrg} \cdots \overset{a_n}{\Rightarrow}_{Mrg} (l_{n+1}, C'_{n+1}) \in \mathit{PathsM}(\mathscr{A})$, and $C_i \subseteq C'_i$, for all $0 \leq i \leq n+1$.

□

We will show in Lemma 10 that the constraint associated to each state in the merged graph is the union of several constraints in the non-merged graph.

The following lemma will be used to prove Lemma 10 by induction. In the proof, we use the following notation: Given a state (l_i, C_i) of $\mathrm{L}(\mathscr{A})$, we write $(l_i, C_i) \overset{a}{\not\Rightarrow}$ if there exists no (l_{i+1}, C_{i+1}) in $\mathrm{L}(\mathscr{A})$ such that $(l_i, C_i) \overset{a}{\Rightarrow} (l_{i+1}, C_{i+1}) \in \Rightarrow(\mathscr{A})$.

Lemma 9. *Let $\mathscr{A}$ be a PTA, let $n \in \mathbb{N}$. Suppose that for all $(l, C) \in \mathit{ReachM}^n(\mathscr{A})$ there exist $m \in \mathbb{N}$ and $(l, C_1), \ldots, (l, C_m) \in \mathit{Reach}^*(\mathscr{A})$ such that*

$$C = \bigcup_{1 \leq i \leq m} C_i.$$

Let $(l, C) \overset{a}{\Rightarrow} (l', C') \in \Rightarrow^n_{Mrg}(\mathscr{A})$. Then there exist $m' \in \mathbb{N}$ and $(l, C'_1), \ldots, (l, C'_l) \in \mathit{Reach}^n(\mathscr{A})$ such that

$$C' = \bigcup_{1 \leq i \leq m'} C'_i.$$

Proof. Suppose $(l, C) \overset{a}{\Rightarrow}_{Mrg} (l', C') \in \Rightarrow^n_{Mrg}(\mathscr{A})$. Suppose (l', C') is the result of merging two states (l', C'_a) and (l', C'_b); hence $C' = C'_a \cup C'_b$. (For sake of simplicity, we assume that only 2 states are merged; the reasoning can be generalized to n states in a straightforward manner.) By definition of the semantics of $\Rightarrow^n_{Mrg}$, there exist C_a and C_b such that $(l_a, C_a) \overset{a}{\Rightarrow} (l', C'_a)$ and $(l_b, C_b) \overset{b}{\Rightarrow} (l', C'_b)$. In our case, we have $(l_a, C_a) = (l, C)$. This is depicted in Fig. 3.2.

From the hypothesis, there exist $m_a \in \mathbb{N}$ and $(l, C^1_a), \ldots, (l, C^{m_a}_a) \in \mathit{Reach}^*(\mathscr{A})$ such that $C_a = \bigcup_{1 \leq i \leq m_a} C^i_a$, and there exist $m_b \in \mathbb{N}$ and $(l_b, C^1_b), \ldots, (l_b, C^{m_b}_b) \in \mathit{Reach}^*(\mathscr{A})$ such that $C_b = \bigcup_{1 \leq i \leq m_b} C^i_b$.

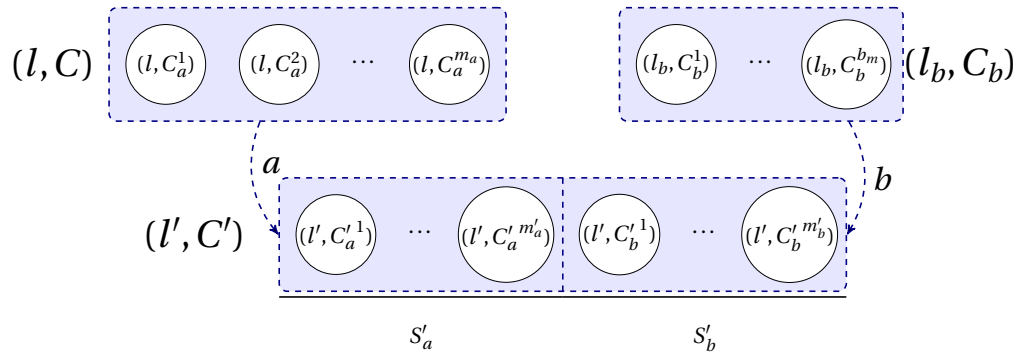


Figure 3.2: Context of Lemma 9

Let $S_a = \{(l,C_a^i) \in \bigcup_{1\leq i\leq m_a}(l,C_a^i) | (l,C_a^i) \overset{a}{\Rightarrow} (l',C_a'^{\,i})$, such as the union of all $C_a'^{\,i}$ is convex$\}$. That is, S_a is the set of states composing the merged state (l,C) that have a successor state through action a, and such that the union of the constraint of their successor state is convex. There may be different such sets of states, with a disjoint convex constraint; for the sake of simplicity, we assume there is only one. Since there is only one, then for each state $s \notin S_a$, we have $s \overset{a}{\not\Rightarrow}$. Let S'_a be the set of target states of S_a. That is, $S'_a = \{(l',C') | \exists (l,C) \in S_a, (l,C) \overset{a}{\Rightarrow} (l',C'_i)\}$. Observe that, since all states in S_a belong to $Reach^*(\mathscr{A})$, then all states in S'_a belong to $Reach^*(\mathscr{A})$ too. Also note that, by construction, states in S'_a can be merged to give only one merged state.

By Lemma 5, we have:

$$(l, \bigcup_{(l,C)\in S_a} C) \overset{a}{\Rightarrow} (l', \bigcup_{(l,C)\in S_a} \{C' | (l,C) \overset{a}{\Rightarrow} (l',C')\})$$

By definition of S'_a, we have:

$$(l, \bigcup_{(l,C)\in S_a} C) \overset{a}{\Rightarrow} (l', \bigcup_{(l',C')\in S'_a} C')$$

Since for each state $s \notin S_a$, we have $s \overset{a}{\not\Rightarrow}$, then

$$(l, \bigcup_{1\leq i\leq m_a} C_i) \overset{a}{\Rightarrow} (l', \bigcup_{(l',C')\in S'_a} C')$$

Since $C_a = \bigcup_{1\leq i\leq m_a} C_i$, then

$$(l,C_a) \overset{a}{\Rightarrow} (l', \bigcup_{(l',C')\in S'_a} C')$$

By following a similar reasoning for C_b, we have

$$(l_b, C_b) \overset{b}{\Rightarrow} (l', \bigcup_{(l',C')\in S'_b} C')$$

with S'_b a set defined in a similar manner to S'_a. Then, we have

$$C'_a = \bigcup_{(l',C')\in S'_a} C' \text{ and } C'_b = \bigcup_{(l',C')\in S'_b} C'$$

Hence

$$C' = \left(\bigcup_{(l',C')\in S'_a} C' \right) \cup \left(\bigcup_{(l',C')\in S'_b} C' \right)$$

By construction, all states of S'_a and S'_b belong to $Reach^*(\mathscr{A})$. This gives the result. □

Lemma 10 (Merging and reachability ($\Longleftarrow$))**.** *Let $\mathscr{A}$ be a PTA. For all $n \in \mathbb{N}$, for all $(l,C) \in ReachM^n(\mathscr{A})$, there exist $m \in \mathbb{N}$ and $(l,C_1),\ldots,(l,C_m) \in Reach^*(\mathscr{A})$ such that*

$$C = \bigcup_{1\le i\le m} C_i.$$

Proof. By induction on n.

- **Base case.** For $n = 0$, we have only one state (l_0, C_0) in $ReachM^n(\mathscr{A})$. By lemma 6, $(l_0, C_0) \in Reach^*(\mathscr{A})$, hence the result holds.
- **Induction step.** Suppose that $n \ge 0$ and the property holds for n. Then, by Lemma 9, the results holds for $n+1$.

□

We can finally characterize the merging in the following theorem.

Theorem 2 (Merging states in PTAs)**.** *Let $\mathscr{A}$ be a PTA. Then:*

1. *For all $(l_0, C_0) \overset{a_0}{\Rightarrow} \cdots \overset{a_{n-1}}{\Rightarrow} (l_n, C_n) \in Paths(\mathscr{A})$, there exist $C'_1,\ldots,C'_n$ such that:*
   (a) $(l_0, C_0) \overset{a_0}{\Rightarrow}_{Mrg} (l_0, C'_1) \overset{a_1}{\Rightarrow}_{Mrg} \cdots \overset{a_{n-1}}{\Rightarrow}_{Mrg} (l_n, C'_n) \in PathsM(\mathscr{A})$, *and*
   (b) $C_i \subseteq C'_i$, *for all* $1 \le i \le n$.
2. *For all $(l,C) \in ReachM^*(\mathscr{A})$ there exist $m \in \mathbb{N}$ and $(l,C_1),\ldots,(l,C_m) \in Reach^*(\mathscr{A})$ such that*

$$C = \bigcup_{1\le i\le m} C_i.$$

Proof. From Lemmas 8 and 10. □

We can derive several results from Theorem 2.

First, each trace in the non-merged graph exists in the merged graph. (Note that the converse statement does not hold.) Hence, $TracesM(\mathscr{A})$ is an over-approximation of $Traces(\mathscr{A})$.

Corollary 3 (Inclusion of traces)**.** *Let $\mathscr{A}$ be a PTA. Then:*

$$Traces(\mathscr{A}) \subseteq TracesM(\mathscr{A}).$$

Proof. From Theorem 2 item 1. □

We state below that each timed-abstract transition in the non-merged graph exists in the merged graph, and vice versa. (Note that this cannot be generalized to complete traces.)

Corollary 4 (Preservation of time-abstract transitions)**.** *Let $\mathscr{A}$ be a PTA. Then:*

1. *Let $l \overset{a}{\Rightarrow} l' \in Traces(\mathscr{A})$. Then $l \overset{a}{\Rightarrow}_{Mrg} l' \in TracesM(\mathscr{A})$.*
2. *Let $l \overset{a}{\Rightarrow}_{Mrg} l' \in TracesM(\mathscr{A})$. Then $l \overset{a}{\Rightarrow} l' \in Traces(\mathscr{A})$.*

Proof. From Theorem 2. □

Finally, locations and actions are preserved by the merging reduction.

Corollary 5 (Preservation of locations and actions)**.** *Let $\mathscr{A}$ be a PTA. Then:*

$$Locations(\mathscr{A}) = LocationsM(\mathscr{A}) \text{ and } Actions(\mathscr{A}) = ActionsM(\mathscr{A}).$$

Proof. From Theorem 2. □

To summarize, computing the set of reachable states using the merging reduction yields an over-approximation of the set of paths. In the original semantics, each trace of $\mathscr{A}(K)$ exists in $\mathscr{A}[\pi]$ for at least one valuation $\pi \models K$; this is not the case anymore with the use of merging, where some traces in $\mathscr{A}(K)$ may not exist for any $\pi \models K$. Nevertheless, both the set of reachable locations and the set of actions are identical to those computed using the original semantics. As a consequence, the merging reduction can be safely used to verify the reachability or the non-reachability of a (set of) location(s), but not to verify more complex properties such as properties on traces (linear-time formulas).

3.3 The Inverse Method with Merging

3.3.1 Principle

We extend *IM* with the merging operation, by merging states within the algorithm, i.e., by replacing within Algorithm 1 all occurrences of *Post* with *PostM*. The extension IM_{Mrg} is given in Algorithm 4.

Algorithm 4: Inverse method with merging $IM_{Mrg}(\mathscr{A}, \pi)$

input : PTA $\mathscr{A}$ of initial state s_0, parameter valuation π
output: Constraint K_{Mrg} over the parameters

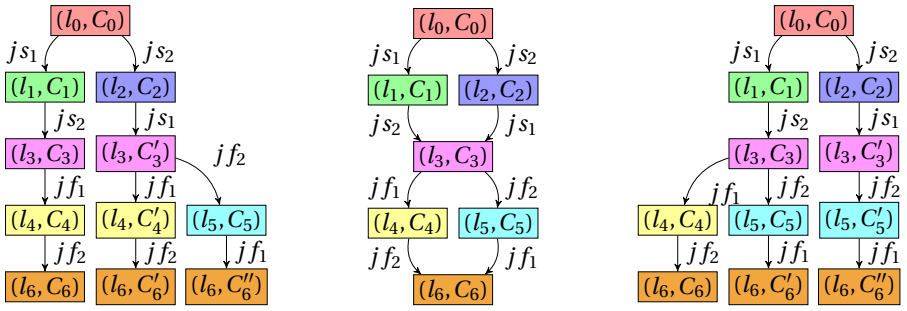
$i \leftarrow 0$; $K_c \leftarrow \texttt{true}$; $S_{new} \leftarrow \{s_0\}$; $S \leftarrow \{\}$
while `true` **do**
  while *there are* π*-incompatible states in* S_{new} **do**
    Select a π-incompatible state (l, C) of S_{new} (i.e., s.t. $\pi \not\models C$) ;
    Select a π-incompatible J in $C{\downarrow_P}$ (i.e., s.t. $\pi \not\models J$) ;
    $K_c \leftarrow K_c \wedge \neg J$; $S \leftarrow \bigcup_{j=0}^{i-1} PostM^{j}_{\mathscr{A}(K_c)}(\{s_0\})$; $S_{new} \leftarrow PostM_{\mathscr{A}(K_c)}(S)$;
  if $S_{new} \sqsubseteq S$ **then return** $K_{Mrg} \leftarrow \bigcap_{(l,C)\in S} C{\downarrow_P}$;
  $i \leftarrow i+1$; $S \leftarrow S \cup S_{new}$; $S_{new} \leftarrow PostM_{\mathscr{A}(K_c)}(S)$

Remark 1. *In* IM_{Mrg}*, states are merged* before *the* π*-compatibility test. Hence, some* π*-incompatible states may possibly be merged, and hence become* π*-compatible. As a consequence, less inequalities will be negated and added to* K_c*, thus giving a weaker output constraint* K_{Mrg}*. Also note that the addition of merging to IM adds a new reason for non-confluence since the merging process is itself non-deterministic.*

We will see that, in contrast to *IM*, IM_{Mrg} does not preserve traces. That is, given $\pi, \pi' \models K_{Mrg}$, a trace in $\mathscr{A}[\pi]$ may not exist in $\mathscr{A}[\pi']$, and vice versa.

Example 11. *We use here a typical jobshop example in the setting of parametric schedulability [FLMS12], in order to show that the traces are no longer preserved with* IM_{Mrg}*. This system (modeled by a PTA* $\mathscr{A}$*) contains 2 machines on which 2 jobs should be performed. The system parameters are* d_i *(for* $i = 1, 2$*) that encode the duration of each job. The system actions are* js_1 *(job 1 starting),* jf_1 *(job 1 finishing) and similarly for job 2.*

Consider $\pi = \{d_1 := 1, d_2 := 2\}$*. The trace set of* $\mathscr{A}[\pi]$ *using the standard semantics (Definition 22) is given in Fig. 3.3(a) (in the form of a graph). Applying*



(a) Trace set of $\mathscr{A}(\pi)$ (b) For $IM_{Mrg}(\mathscr{A}, \pi)$ (c) Trace set of $\mathscr{A}(\pi')$

Figure 3.3: Trace sets of $\mathscr{A}$

IM to $\mathscr{A}$ and π gives $K = d_2 > d_1$. From the correctness of IM [AS13], the trace set of $\mathscr{A}[\pi']$, for all $\pi' \models K$, is the same as for $\mathscr{A}[\pi]$. Now, applying IM_{Mrg} to $\mathscr{A}$ and π gives $K_{Mrg} = \texttt{true}$. We depict in Figure 3.3(b) an abstract representation of all possible trace sets, for all $\pi \models K_{Mrg}$. K_{Mrg} has been obtained with IM_{Mrg}. (l_3, C_3) and (l_6, C_6) have been obtained by the merging of concrete states. Then, let $\pi' = \{d_1 := 2, d_2 := 1\}$ be a valuation in K_{Mrg} but outside of K. The trace set of $\mathscr{A}[\pi']$ (using the standard semantics) is given in Fig. 3.3(c). The trace sets of $\mathscr{A}[\pi]$ and $\mathscr{A}[\pi']$ are different: the trace $l_0 \overset{js_2}{\Rightarrow} l_2 \overset{js_1}{\Rightarrow} l_3 \overset{jf_1}{\Rightarrow} l_4 \overset{jf_2}{\Rightarrow} l_6$ exists in $\mathscr{A}[\pi]$ but not in $\mathscr{A}[\pi']$; the trace $l_0 \overset{js_1}{\Rightarrow} l_1 \overset{js_2}{\Rightarrow} l_3 \overset{jf_2}{\Rightarrow} l_5 \overset{jf_1}{\Rightarrow} l_6$ exists in $\mathscr{A}[\pi']$ but not in $\mathscr{A}[\pi]$. However, note that the reachable locations and executable actions are the same in these two trace sets.

3.3.2 Preservation of Locations

We will show in Theorem 6 that IM_{Mrg} preserves locations. This result relies on the forthcoming lemma.

Lemma 11. *Suppose $IM_{Mrg}(\mathscr{A}, \pi)$ terminates with output K_{Mrg}. Then $\pi \models K_{Mrg}$.*

Proof. At the end of IM_{Mrg}, all merged states in S are π-compatible by construction. That is, for all $(l, C) \in S$, we have $\pi \models C{\downarrow_P}$. Since $K_{Mrg} = \bigcap_{(l,C)\in S} C{\downarrow_P}$, then $\pi \models K_{Mrg}$. □

Theorem 6. *Suppose $IM_{Mrg}(\mathscr{A}, \pi)$ terminates with output K_{Mrg}. Then, for all $\pi' \models K_{Mrg}$, $Locations(\mathscr{A}[\pi]) = Locations(\mathscr{A}[\pi'])$.*

Proof. From Lemma 11, we have that $\pi \models K_{Mrg}$. We will first show that $Locations(\mathscr{A}[\pi]) = Locations(\mathscr{A}(K_c))$, where K_c denotes the value of the current constraint at the end of the algorithm.

$\Longrightarrow$ The fact that $\mathit{Locations}(\mathscr{A}[\pi]) \subseteq \mathit{Locations}(\mathscr{A}(K_c))$ is straightforward. First note that $K_{Mrg} \subseteq K_c$. Hence $\pi \models K_{Mrg}$ implies that $\pi \models K_c$. Hence any location in $\mathscr{A}[\pi]$ is reachable in $\mathscr{A}(K_c)$.

$\Longleftarrow$ We now show that $\mathit{Locations}(\mathscr{A}(K_c)) \subseteq \mathit{Locations}(\mathscr{A}[\pi])$. Let us pick up a state $(l, C) \in S$, where S is the set of states at the end of the algorithm. At the end of the algorithm, $S = \mathit{ReachM}^*(\mathscr{A}(K_c))$. Hence $l \in \mathit{LocationsM}(\mathscr{A}(K_c))$. From Corollary 5, we have that $\mathit{Locations}(\mathscr{A}(K_c)) = \mathit{LocationsM}(\mathscr{A}(K_c))$. Hence $l \in \mathit{Locations}(\mathscr{A}(K_c))$.

We will show that l belongs to $\mathit{Locations}(\mathscr{A}[\pi])$. Recall that $K_{Mrg} = \bigcap_{(l,C)\in S} C{\downarrow_P}$. Since $\pi \models K_{Mrg}$ then, for all $(l, C) \in S$, it holds that $\pi \models C{\downarrow_P}$. hence this is also the case for the (l, C) we picked up. From Theorem 2, there exist $m \in \mathbb{N}$ and $(l, C_1), \ldots, (l, C_m) \in \mathit{Reach}^*(\mathscr{A}(K_c))$ such that

$$C = \bigcup_{1 \leq i \leq m} C_i.$$

Let us pick a constraint C_i such that $\pi \models C_i{\downarrow_P}$. Such a C_i necessarily exists since $\pi \models C{\downarrow_P}$.

Now, we can apply classical results on PTAs: consider a path of $\mathscr{A}(K_c)$ reaching (l, C_i); then, since $\pi \models C_i{\downarrow_P}$, there exists an equivalent path in $\mathscr{A}[\pi]$ (see, e.g., [AS13, Proposition 18]). Hence $l \in \mathit{Locations}(\mathscr{A}[\pi])$.

Now, let π' be a valuation such that $\pi' \models K_{Mrg}$. Using the same reasoning, it holds that $\mathit{Locations}(\mathscr{A}[\pi']) = \mathit{Locations}(\mathscr{A}(K_c))$.

Hence $\mathit{Locations}(\mathscr{A}[\pi]) = \mathit{Locations}(\mathscr{A}[\pi'])$. □

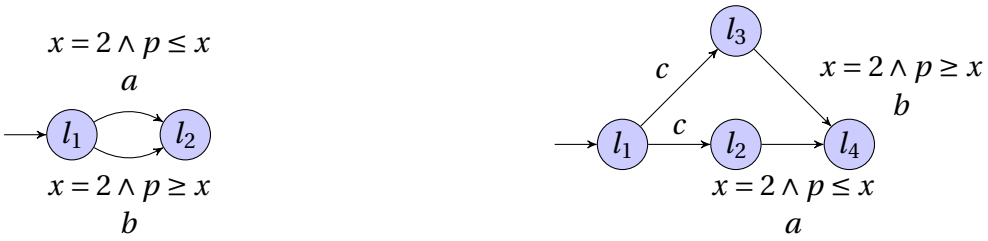
Hence, although the trace set is not preserved by IM_{Mrg}, the set of locations is. As a consequence, the reachability and safety properties (based on locations) that are true in $\mathscr{A}[\pi]$ are also true in $\mathscr{A}[\pi']$.

3.3.3 Preserving Actions

General Case

Although the set of locations is preserved by IM_{Mrg}, the set of actions is not preserved in the general case (in contrast to the reachability analysis with merging).

Consider the simple PTA in Fig. 3.4(a). Observe that action a (respectively b) can be taken only if $p \leq 2$ (respectively $p \geq 2$). For a reference valuation π such that $p = 1$, only a can be executed. On the one hand, IM applied to this PTA and to π will output constraint $p < 2$, implying that only a can be executed.



(a) PTA 1 (b) PTA 2

Figure 3.4: Counterexample PTAs showing the non-preservation of actions by IM_{Mrg}

On the other hand, IM_{Mrg} will reach two symbolic states $(l_2, p \leq 2 \wedge x \geq 2)$ and $(l_2, p \geq 2 \wedge x \geq 2)$, and will merge them into $(l_2, p \geq 0 \wedge x \geq 2)$. Since this state is π-compatible, IM_{Mrg} will output `true` as a constraint. Then, choosing a valuation π' such as $p = 3$ (note that $\pi' \models IM_{Mrg}(\mathscr{A}, \pi)$), only b can be executed in $\mathscr{A}[\pi']$. Hence the set of actions is not the same as in $\mathscr{A}[\pi]$.

A restriction to the model so that IM_{Mrg} preserves actions could have been that, for each couple of locations l_1 and l_2, all transitions from l_1 to l_2 are always firing the same given action a. But this is not enough: the PTA in Fig. 3.4(b) conforms to this restriction, but the same situation as in Fig. 3.4(a) happens.

Proposition 3 (Non-preservation of actions)**.** *There exist $\mathscr{A}$, π and π' such that (1) $IM_{Mrg}(\mathscr{A}, \pi)$ terminates with output K_{Mrg}, (2) $\pi' \models K_{Mrg}$, and (3) $Actions(\mathscr{A}[\pi]) \neq Actions(\mathscr{A}[\pi'])$.*

Not all properties are based on actions. Hence IM_{Mrg} is suitable for systems the correctness of which is expressed using the reachability or the non-reachability of locations. Nevertheless, to be able to handle as well systems the correctness of which is expressed using the (non-)reachability of actions, the rest of this section will be devoted to identifying techniques to preserve actions too.

Backward-Deterministic Parametric Timed Automata

We identify here a subclass of PTAs for which IM_{Mrg} preserves the set of actions. We restrict the model so that, for any location, at most one action is used on its incoming edges. This restriction can be checked syntactically.

Definition 29 (Backward-determinism)**.** *A PTA $\mathscr{A}$ is said to be* backward-deterministic *if for all $(l_1, g, a, \rho, l_2), (l_1', g', a', \rho', l_2') \in \rightarrow$, then $l_2 = l_2' \Longrightarrow a = a'$.*

In a backward-deterministic PTA, if a location is reachable, then its incoming action is executed too. Hence the preservation of the locations by IM_{Mrg} implies the preservation of the actions too.

Proposition 4 (Action preservation)**.** *Let $\mathscr{A}$ be a backward-deterministic PTA. Suppose $IM_{Mrg}(\mathscr{A},\pi)$ terminates with output K_{Mrg}. Then, for all $\pi' \models K_{Mrg}$, $Actions(\mathscr{A}[\pi]) = Actions(\mathscr{A}[\pi'])$.*

Proof. From Theorem 6 and Definition 29. □

This restriction of backward-determinism may be seen as quite strong in practice. Hence, in the following, in order to preserve the set of actions, we propose to modify the algorithm itself rather than restricting the model.

Improvement of the Inverse Method

The non-preservation of the actions by IM_{Mrg} comes from the fact that the states are first merged, and then tested against π-compatibility (see Remark 1). In order to guarantee the action preservation, we propose to first test newly generated states against π-compatibility, and then merge them. Although this modification is only a subtle inversion of two operations in the algorithm, it has consequences on the properties preserved by the new algorithm.

We introduce an improved version IM'_{Mrg} of IM_{Mrg} in Algorithm 5, where states are merged after the π-compatibility tests. Technically, the differences with IM_{Mrg} (highlighted using a non-white background) are as follows: (1) the operation to compute the states at the current deepest level i is *Post* instead of *PostM* (lines 9 and 6), and (2) the states are merged *after* the end of the π-incompatibility tests (addition of line 7).

We classify below the constraints output by the 3 versions of IM.

Proposition 5. *Suppose $IM(\mathscr{A},\pi)$, $IM_{Mrg}(\mathscr{A},\pi)$ and $IM'_{Mrg}(\mathscr{A},\pi)$ terminate in a deterministic manner with an output K, K_{Mrg} and K'_{Mrg}, respectively.*

Then, $K \subseteq K'_{Mrg} \subseteq K_{Mrg}$

Proof. The first part of the relationship ($K \subseteq K'_{Mrg}$) comes from two reasons.

1. First, less inequalities are negated. Consider the case of two states s_1 and s_2 (both π-compatible) at a level n, such that each of them has a successor at level $n+1$, one π-compatible (say s'_1), the other one π-incompatible (say s'_2). In IM, the second state s'_2 would be removed by negating a π-incompatible inequality J. Now, if these two states s_1 and s_2 are merged at level n in IM'_{Mrg} (giving state, say s), their merging could give only one

Algorithm 5: Inverse method with merging (variant) $IM'_{Mrg}(\mathscr{A},\pi)$

input : PTA $\mathscr{A}$ of initial state s_0, parameter valuation π
output: Constraint K'_{Mrg} over the parameters

$i \leftarrow 0$; $K_c \leftarrow \texttt{true}$; $S_{new} \leftarrow \{s_0\}$; $S \leftarrow \{\}$
while `true` **do**
  while *there are π-incompatible states in S_{new}* **do**
    Select a π-incompatible state (l,C) of S_{new} (i.e., s.t. $\pi \not\models C$) ;
    Select a π-incompatible J in $C{\downarrow_P}$ (i.e., s.t. $\pi \not\models J$) ;
    $K_c \leftarrow K_c \wedge \neg J$; $S \leftarrow \bigcup_{j=0}^{i-1} PostM^j_{\mathscr{A}(K_c)}(\{s_0\})$; $S_{new} \leftarrow Post_{\mathscr{A}(K_c)}(S)$;
  $S_{new} \leftarrow Merge(S_{new})$
  if $S_{new} \sqsubseteq S$ **then return** $K'_{Mrg} \leftarrow \bigcap_{(l,C)\in S} C{\downarrow_P}$;
  $i \leftarrow i+1$; $S \leftarrow S \cup S_{new}$; $S_{new} \leftarrow Post_{\mathscr{A}(K_c)}(S)$

successor s' (equivalent to the union of s'_1 and s'_2); if this successor is π-compatible, the inequality J will not be found, hence not be negated. As a consequence, IM'_{Mrg} will meet less π-incompatible states than IM.

2. The second reason is that the final intersection of the constraints over the parameters associated with all reachable states will be weaker in the case of IM'_{Mrg}. Indeed, since states have been merged, we will consider their union instead of their intersection. Considering the situation in Fig. 3.2, IM will return the intersection of all local constraints, whereas IM'_{Mrg} will only output $C{\downarrow_P} \cap C_b{\downarrow_P} \cap C'{\downarrow_P}$.

The second part of the relationship ($K'_{Mrg} \subseteq K_{Mrg}$) comes from a similar reasoning. In short, less π-incompatible inequalities will be negated, since the merging is performed before the π-compatibility test. This situation is depicted in Fig. 3.4(a). And similarly, since states are more often merged, their final intersection will yield a weaker constraint too. □

Note that IM'_{Mrg} still does not preserve traces; the situation in Fig. 3.3 is exactly the same for IM'_{Mrg} as for IM_{Mrg}.

Theorem 7. *Suppose $IM'_{Mrg}(\mathscr{A},\pi)$ terminates with output K'_{Mrg}. Then, for all $\pi' \models K'_{Mrg}$:*

1. *$Locations(\mathscr{A}[\pi]) = Locations(\mathscr{A}[\pi'])$, and*
2. *$Actions(\mathscr{A}[\pi]) = Actions(\mathscr{A}[\pi'])$.*

Proof. Preservation of locations follows the same reasoning as for Theorem 6. Preservation of actions is guaranteed by construction of IM'_{Mrg} together with the preservation of locations. □

3.4 Experimental Validation

We implemented IM'_{Mrg} in IMITATOR [AFKS12], in addition to the classical *IM*. In [Dav05], the main technique for merging two timed constraints C, C' consists in comparing their convex hull H with their union. If the hull and the union are equal (or alternatively, if $(H \setminus C) \setminus C' = \emptyset$, where $\setminus$ is the operation of *convex difference*), then C and C' are mergeable into H. In [Dav05, Dav06], this technique is specialized to the case where the timed constraints are represented as DBMs. DBMs are not suitable to represent the state space of PTAs; in IMITATOR, polyhedra are used. We implemented the mergeability test using the (costly) operation of convex merging from the Parma Polyhedra Library (PPL) [BHZ08].

Table 3.1 describes experiments comparing the performances and results of *IM* and IM'_{Mrg}. Column $|X|$ (resp. $|P|$) denotes the number of clocks (resp. parameters) of the PTA. For each algorithm, columns States, Trans., M, t and Cpl denote the number of states, of transitions, a tentative size of the heap given by OCaml, the computation time in seconds, and whether the resulting constraint is complete[1], respectively. In the last 3 columns, we compare the results: first, we divide the number of states in *IM* by the number of states in IM'_{Mrg} and multiply by 100 (hence, a number smaller than 100 denotes an improvement of IM'_{Mrg}); second, we perform the same comparison for the computation time; the last column indicates whether $K = K'_{Mrg}$ or $K \subsetneq K'_{Mrg}$. Experiments were performed on a KUbuntu 12.10 64 bits system running on an Intel Core i7 CPU 2.67GHz with 4 GiB of RAM.

The first 4 models are asynchronous circuits [CC07, AS13]. The SIMOP case study is an industrial networked automation system [AS13]. The next 5 models are common protocols [DKRT97, HRSV02, AS13]. The other models are scheduling problems (e.g., [AM02, BB04a, LPP+10]). All models are described and available (with sources and binaries of IMITATOR) on IMITATOR's Web page[2].

From Table 3.1, we see that IM'_{Mrg} has the following advantages. First, the state space is often reduced (actually, in all but 4 models) compared to *IM*. This is particularly interesting for the scheduling problems, with a division of the number of states by a factor of up to 16 (LA02). Also note that two case

[1] Whereas *IM* and IM'_{Mrg} may be non-complete in the general case, IMITATOR exploits a sufficient (but non-necessary) condition for completeness to detect completeness, when possible.

[2] http://www.lsv.ens-cachan.fr/Software/imitator/merging/

			IM					IM'_{Mrg}					Comparison		
Example	$\|X\|$	$\|P\|$	States	Trans.	t	M	Cpl	States	Trans.	t	M	Cpl	States	t	K
AndOr	4	12	11	11	0.052	1.98	√	9	9	0.056	1.97	√	82	108	=
Flip-Flop	5	12	11	10	0.060	2.75	√	9	9	0.057	2.74	√	82	108	=
Latch	8	13	18	17	0.083	2.65	?	12	12	0.069	2.51	?	67	83	=
SPSMALL	10	26	31	30	0.618	7.746	?	31	30	0.613	8.254	?	100	99	=
SIMOP	8	7	-	-	-	OoM	-	172	262	2.52	35.3	?	0	0	-
BRP	7	6	429	474	3.50	40.0	√	426	473	4.30	41.1	√	99	123	=
CSMA/CD	3	3	301	462	0.514	12.5	√	300	461	0.574	13.2	√	100	112	=
CSMA/CD'	3	3	13,365	14,271	18.3	695	√	13,365	14,271	25.4	739	√	100	139	=
RCP	5	6	327	518	0.748	17.5	√	115	186	0.684	22.3	√	35	91	=
WLAN	2	8	-	-	-	OoM	-	8,430	15,152	2,137	100,502	√	0	0	-
ABT	7	7	63	62	0.344	7.55	?	63	62	0.335	7.44	?	100	97	=
AM02	3	4	182	215	0.369	6.18	√	53	70	0.112	3.49	√	29	30	$\subsetneq$
BB04	6	7	806	827	28.0	66.4	?	141	145	3.15	14.8	?	17	11	=
CTC	15	21	1,364	1,363	88.9	150	√	215	264	17.6	27.6	√	16	20	=
LA02	3	5	6,290	8,023	751	425	?	383	533	17.7	44.6	√	6.0	2.4	$\subsetneq$
LPPRC10	4	7	78	102	0.39	5.83	?	31	40	0.251	3.56	?	40	64	=
M2.4	3	8	1,497	1,844	8.89	95.7	√	119	181	0.374	9.12	√	7.9	4.2	$\subsetneq$

Table 3.1: Comparison between IM and IM'_{Mrg}

studies could not even be verified without the merging reduction, due to memory exhaustion ("OoM"). Second, the computation time is almost always reduced when the merging reduction indeed reduces the state space, by a factor of up to 42 (LA02). Third, and more surprisingly (considering the cost of the mergeability test), the overhead induced by the mergeability test often does not yield a significant augmentation of the computation time, even when the merging reduction does not reduce the state space at all; the worst case is +39 % (CSMA/CD'), which remains reasonable. Finally, the constraint output by IM'_{Mrg} is weaker (i.e., corresponds to a larger set of valuations) than IM for some case studies (namely, scheduling problems).

3.5 Discussion

We have shown in this chapter that (1) a general technique of state merging in PTAs preserves both the reachability and the non-reachability of actions and locations, (2) the integration of this technique into IM often synthesizes a weaker (hence, better) constraint while reducing the computation space, and preserves locations (but neither traces nor actions), and (3) an improved version of IM_{Mrg} preserves not only locations but actions. Experiments with IMITATOR show that the improved procedure IM'_{Mrg} does not only reduce the state space, but is also often faster than the original procedure IM.

As future work, it would be interesting to to study the combined integration into IM of the general technique of state merging with variants [AS11] and optimizations [And13b] of IM. Regarding the implementation in IMITATOR, we aim at studying the replacement of polyhedra with parametric DBMs [HRSV02]; furthermore, the (costly) mergeability test should be optimized so as to improve

performance. Finally, we also plan to generalize the merging technique to the hybrid setting [FK13].

3.6 Related Work

In [SBM06], it is shown that, in a network of TAs, all the successor states can be merged together when all the interleavings of actions are possible. However, this result does not extend to the parametric case In [Dav05, Dav06], it is proposed to replace the union of two states by a unique state when the union of their continuous part (viz., the symbolic clock values) is convex, and the discrete part (viz., the location) is identical. More precisely, if the union of the continuous part of two states is included into their convex hull, then the two states can be replaced with their hull. This technique is applied to timed constraints represented in the form of Difference Bound Matrices (DBMs). Our merging technique can be seen as an extension of the technique in [Dav05, Dav06] to the parametric case. This extension is not trivial (for example, the merging technique of [SBM06] does not extend to the parametric case), and the implementation is necessarily different, since DBMs (in their original form) do not allow the use of parameters. Instead, we implemented our approach in IMITATOR using polyhedra [BHZ08].

Chapter 4

Application to the Robustness Analysis of Scheduling Problems

In this chapter, we use the inverse method for timed automata recalled in Chapter 2 to analyze specifically the robustness of real-time scheduling systems. Furthermore, we use the behavioral cartography of Section 2.7 to synthesize schedulable zones of real-time systems. More precisely, we are interested here in representing and analyzing the *schedulability region*, i.e., the region of parameter space that corresponds to a feasible design.

Outline of the Chapter Basis scheduling definitions are introduced in Section 4.1. In Section 4.2, we explain on an example the principle of the application of the inverse method to scheduling problems. In Section 4.3, we apply the method to various schedulability problems of the literature (jobs with variable execution times, deadlines), as well as to an industrial case study. An industrial case study is treated in Section 4.3.3. We compare our approach with an analytic method in Section 4.4. The results are discussed in Section 4.5. Related work is discussed in Section 4.6.

4.1 Preliminaries

4.1.1 Scheduling Problems

A real-time system $\mathscr{S}$ is viewed in this chapter as a set of jobs $\{J_1, J_2, \ldots, J_n\}$. A job J_i generates a possibly infinite stream of tasks $\{J_{i,1}, J_{i,2}, \ldots\}$. When a job is activated, it executes for at most time C_i, and has to terminate within a relative deadline D_i. Some real-time systems feature a *preemption* mechanism, described as follows. Tasks may have a different priority; when a task of low pri-

ority is preempted by a task with higher priority, the task with lower priority is interrupted, and will be resumed once the task of higher priority is completed. The activation of tasks can be modeled by parametric timed automata, where activation events are associated with transition labels. The timings (C_i, D_i) can be considered as parameters associated with each job. A *parametric job system* $\mathscr{S}$ is a set $\{J_1, J_2, \ldots, J_n\}$ associated with a vector P of parameters. Each design parameter in P can have a fixed (known) constant value or be a free parameter (i.e., an unknown constant).

Given a reference valuation π, an *instantiated* job system $\mathscr{S}[\pi]$ is a job system $\{J_1, J_2, \ldots, J_n\}$ associated with a vector of design parameter P where each design parameter in P is assigned a fixed value according to the valuation π. For a given choice π of parameters, we say that a job J_i is *schedulable* if all the generated tasks $J_{i,k}$ finish their execution before the deadline. The system $\mathscr{S}[\pi]$ is *schedulable* if all its jobs are schedulable. In this context, the problem of *robustness* is defined as follows.

Problem 1. *Given a parametric job system $\mathscr{S}$ and a valuation π_0 of the parameters, find a constraint K_0 containing π_0 such that $\mathscr{S}$ is* robust on K_0*, i.e., for all $\pi \models K_0$, $\mathscr{S}[\pi]$ is schedulable if and only if $\mathscr{S}[\pi_0]$ is schedulable.*

We are also interested in the following problem of computation of schedulability zones. This problem can be seen as a problem equivalent to the good parameters problem in the setting of scheduling problems.

Problem 2. *Consider a parametric job system $\mathscr{S}$, and a rectangle V_0 inside the parameter space. Find the* schedulability zone $\mathscr{Z}$*, defined as the largest subset of valuations π of V_0 for which $\mathscr{S}[\pi]$ is schedulable.*

We show in this chapter that Problem 1 can be solved using the inverse method for parametric timed automata, and Problem 2 can be solved using the behavioral cartography of parametric timed automata.

4.1.2 Timed Automata Augmented with Stopwatches

We informally extend here the definition of timed automata (Definition 8) to the case of stopwatches (see, e.g., [AM02]). Stopwatches are special clocks that can be *stopped* in some locations. This formalism of timed automata equipped with stopwatches (sometimes referred to as *stopwatch automata*) is often used in practice, but many problems decidable for timed automata turn undecidable for timed automata with stopwatches. Timed automata with stopwatches are very useful for modeling scheduling problems with preemption, as shown in [AM02]. We will use in this chapter an extension of the inverse method

to (parameteric) timed automata with stopwatches. The inverse method has been extended in [FK13] to hybrid systems; timed automata with stopwatches are actually a subclass of (linear) hybrid automata, where the clock derivatives used in activities can be either 0 or 1. Therefore, the correctness and properties of *IM* for timed automata with stopwatches can be directly derived from those of *IMH*.

4.1.3 System Model

We consider distributed real-time systems consisting of several computational nodes, each one hosting one single processor, which are connected by one or more shared networks. Without loss of generality, from now on we will use the term *task* to denote both tasks and messages, and the term *processor* to denote both processors and networks.

A distributed real-time system consists of a set of task pipelines $\{\mathscr{P}^1,\ldots,\mathscr{P}^n\}$ to be executed on a set of processors. A *pipeline* is a chain of tasks $\mathscr{P}^j = \{\tau_1^j,\ldots,\tau_n^j\}$ to be executed in order, and each task is allocated on one (possibly different) processor. In order to simplify the notation, in the following we sometimes drop the pipeline superscript when there is no possibility of misinterpretation.

A pipeline is assigned two fixed parameters: T^j is the pipeline period and D^j_{e2e} is the end-to-end deadline. This means that all tasks of the pipeline are activated together every T^j units of time; and all tasks should be completed within a time interval of D^j_{e2e}.

A task in the pipeline can be a piece of code to be executed on a processor or a message to be sent over a network. More precisely, a real-time periodic task is a tuple $\tau_i = (C_i, T_i, D_i, R_i, q_i, p_i, J_i)$.

This task model contains the following fixed parameters:

- T_i is the task period. All tasks in the same pipeline have period equal to the pipeline period T;
- D_i is the task relative deadline;
- q_i is the task priority; the larger q_i, the higher the priority;
- p_i is the index of the processor (or network) on which the task executes.

Also, a task is characterised by the following free parameters (variables):

- C_i is the worst-case computation time (or worst-case transmission time, in case it models a message). It is the worst-case time the task needs to

complete one periodic instance when executed alone on a dedicated processor (or network). In this paper we want to characterise the schedulability of a distributed system in the space of the computation times, so C_i is a free parameter.

- R_i is the *task worst-case response time*, i.e. the worst case finishing time of any task instance relative to the activation of its pipeline.

- J_i is the task worst-case activation jitter, i.e. the greatest time since its activation that a task must wait for all preceding tasks to complete their execution.

Every task activation is an *instance* (or *job*) of the task. We denote the kth instance of task τ_i as $\tau_{i,k}$. An instance $\tau_{i,k}$ of a task in the pipeline can start executing only after the corresponding instance of the preceding task $\tau_{i-1,k}$ has completed. Finally, the last task in the pipeline must complete every instance before D_{e2e} units of time from its pipeline's activation. For a job $\tau_{i,k}$ we define the following notation:

- $a_{i,k}$ is $\tau_{i,k}$'s arrival time (coincident with the activation time of the pipeline).

- $s_{i,k}$ is the start time of the instance, i.e. the first time the instance executes on the processor.

- $f_{i,k}$ is the job's finishing time.

- $r_{i,k}$ the task release time. The first task of a pipeline is released immediately at the time of its arrival $r_{0,k} = a_{0,k}$; successive tasks are released at the finishing time of the preceding tasks: $r_{i,k} = f_{i-1,k}$. The following relationship holds: $\forall i,k \;\; a_{0,k} = a_{i,k} \leq r_{i,k} \leq s_{i,k} < f_{i,k}$

- The maximum difference between arrival and release time is the worst-case activation jitter of the task: $J_i = \max_k(r_{i,k} - a_{i,k})$.

- The maximum difference between finishing time and arrival time is the worst-case response time of the task: $R_i = \max_k(f_{i,k} - a_{i,k})$.

Parameters R_i and J_i depend on the other tasks parameters and on the scheduling policy according to a complex set of equations. Of course, they cannot be considered parameters that the programmer can modify: nevertheless, for our purposes it is useful to consider them as variables to help us write the set of constraints that define the schedulability space (the exact role of such variables will be detailed in Section 4.4.2)..

A scheduling algorithm is *fully preemptive* if the execution of a lower priority job can be suspended at any instant by the release of a higher priority job, which is then executed in its place. A scheduling algorithm is *non-preemptive* if a lower priority job, once it has started executing, can complete its execution regardless of the release of higher priority jobs. In this paper, we consider fully preemptive fixed priority scheduling for processors, and non-preemptive fixed priority scheduling for networks.

Modelling the System Using Parametric Stopwatch Automata

Timed Automata with Stopwatches have been used for modelling scheduling problems in the past. Our model technique is similar to [AM02, AAM06], except that we model pipelines of tasks, and that we use PSA for obtaining the space of feasible computation times. In the current implementation, we only model pipelines with end-to-end deadlines no larger than their periods. This allows us to simplify the model and reduce the complexity of the analysis. The extension to deadlines larger than the period is discussed at the end of the section.

We illustrate our model with the help of an example of two pipelines $\mathscr{P}^1, \mathscr{P}^2$ with $\mathscr{P}^1 = \{\tau_1, \tau_2\}$, $\mathscr{P}^2 = \{\tau_3, \tau_4\}$, $p(\tau_1) = p(\tau_4) = p_1$, $p(\tau_2) = p(\tau_3) = p_2$, p_1 being a preemptive processor and p_2 being non-preemptive. We have that $q_1 > q_4$ and $q_3 > q_2$.

Figure 4.1 shows the PSA model of a pipeline. A pipeline is a sequence of tasks that are to be executed in order: when a task completes its instance, it instantly releases the next task in the pipeline. Since we assume constrained deadlines, once every task in the pipeline has completed, the pipeline waits for the next period to start. This PSA contains one local clock $x_{\mathscr{P}^1}$, one parameter T_1 (the pipeline's period), and synchronises on 5 actions: "τ_1 release", "τ_1 completed", "τ_2 release", "τ_2 completed", and "$\mathscr{P}^1$ restart". The order of these events imposes that task τ_1 must be entirely executed before task τ_2. The initialisation of the pipeline's local clock $x_{\mathscr{P}^1}$ and the invariant $x_{\mathscr{P}^1} \leq T_1$ ensure that the pipeline's execution terminates within its period T_1. The guard $x_{\mathscr{P}^1} == T_1$ ensures that the pipeline restarts after exactly T_1 units of time.

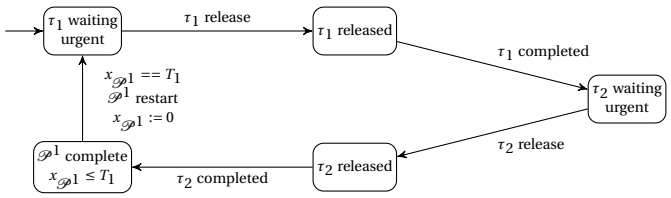


Figure 4.1: PSA modelling a pipeline $\mathscr{P}^1$ with two tasks τ_1, τ_2

Figure 4.2 shows the model of a preemptive processor with 2 tasks τ_1 and τ_4,

where task τ_1 has higher priority over task τ_4. The processor starts by being *idle*, waiting for a task release. As soon as a request has been received (e.g. action "τ_4 release"), it moves to one of the states where the corresponding task is running ("τ_4 running"). If it receives another release request ("τ_1 release"), it moves to the state corresponding to the higher priority task running ("τ_1 release, τ_4 released"). The fact that τ_1 does not execute anymore is modelled by the blocking of the clock x_{τ_4} corresponding to task τ_4. Moreover, while a task executes, the scheduler automaton checks if the corresponding pipeline misses its deadline (e.g. guard $x_{\mathscr{P}^1} > D^1_{e2e}$, where D^1_{e2e} is τ_1's deadline). In the case of a deadline miss, the processor moves to a special failure state ("deadline missed") and stops any further computation.

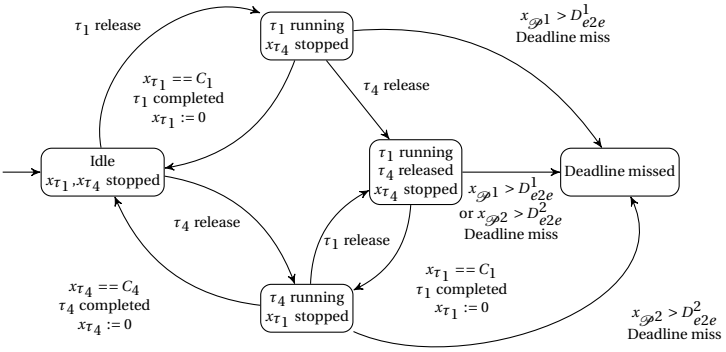


Figure 4.2: PSA modelling a preemptive processor with two tasks τ_1, τ_4

The model of a non-preemptive processor is very similar to the model of preemptive processor: the central state in Figure 4.2 which accounts for the fact that τ_4 is stopped when τ_1 is released, in the non-preemptive case must not stop τ_4, but simply remember that τ_1 has been released, so that we can move to the top state when τ_4 completes its instance.

We use the IMITATOR software tool [AFKS12] implementing the behavioural cartography, to perform the analysis of the PSA. The tool takes as input a textual description of the PSA and an interval of values for each parameter, which can be seen as a hypercube in $|P|$ dimensions, with $|P|$ the number of parameters. Then, it explores the hypercube of values using *IM*, and it outputs a set of tiles.

For each tile, IMITATOR derives whether the corresponding system behaviour is valid (i.e. no deadline miss is present), which corresponds to a good tile, or invalid (at least one deadline miss has been found), which corresponds to a bad tile. Every behaviour can be regarded as a set of traces of the system. Although deadline misses are *timed* behaviours, they are reduced to (untimed) traces thanks to the "deadline miss" location of the processor PSA. All points

inside one particular tile are values of the parameters that generate equivalent behaviours (they correspond to the same trace set).

The result of the behavioural cartography is a set of tiles that covers "almost"[1] the entire hypercube. The region of space we are looking for is the union of all the good tiles.

The proposed model can be extended to deal with deadlines greater than periods by changing the automaton in Figure 4.1. In particular, we must take into account that each task can have up to $\left\lceil \frac{D_{e2e}}{T} \right\rceil$ pending instances that have not completed yet. However, the number of locations increases with $\left\lceil \frac{D_{e2e}}{T} \right\rceil$ and thus the complexity of the analysis.

4.2 Scheduling Analysis Using the Inverse Method

We introduce here a method based on the inverse method, that performs scheduling analysis for real-time systems [FLMS12]. Throughout this section, we explain the method on a preemptive jobshop example introduced in [AM02]. This example is a preemptive scheduling problem, encoded in [AM02] using timed automata augmented with stopwatches. The jobshop scheduling problem is a generic resource allocation problem in which common resources ("machines") are required at various time points (and for given duration) by different tasks. Suppose we are given a fixed set M of machines. A *step* is a pair (m,d) where $m \in M$ and $d \in \mathbb{N}$, indicating the required utilization of resource m for time duration d. A *job* is a finite sequence $J = (m_1,d_1),(m_2,d_2),\cdots,(m_k,d_k)$ of steps stating that in order to accomplish job J, one needs to use a machine m_1 for d_1 time units, then use machine m_2 for d_2 time units, and so on.

4.2.1 Modeling Schedulability with Timed Automata

Consider the jobshop system $\mathscr{S} = \{J_1, J_2\}$ for 2 jobs and 3 machines m_1, m_2, m_3 with: $J_1 = (m_1,d_1),(m_2,d_2),(m_3,d_3)$ and $J_2 = (m_2,d_2')$ with $d_1 = 3, d_2 = 2, d_3 = 4, d_2' = 5$. The classical problem, called the "makespan" problem, consists of finding the minimum time (makespan) needed for completing all the tasks (with the constraint that, at any time, a machine can execute only one task). In [AM02], it is shown how to solve the makespan problem for $\mathscr{S}$ using a timed automaton $\mathscr{A}$. Actually, we do *not* adress the makespan problem here, but rather

[1] Technically, a part might be non-covered in some cases at the border between the good and the bad subspace; this part has a width of at most ϵ, where ϵ is an input of the tool; of course, the smaller ϵ, the more costly the analysis (see [AF10, AS13]).

the *schedulability* problem. More precisely, we suppose that a certain constant bound (or deadline) μ is given, and we ask whether or not the system is *schedulable*, i.e., if there is a way (schedule) to complete all the jobs within μ time units.

In order to treat schedulability, we add to the system an extra "observer" timed automaton (see, e.g., [ABL98, ABBL98, And13a]), that features a special clock measuring the global time. When this special clock goes beyond the deadline μ, the observer goes into a special location called "FAILURE". Hence, a trace ending in location "FAILURE" is a bad trace, because the deadline has been fired before all jobs are completed. On the contrary, if all jobs could be completed before the deadline μ, the observer goes into a special location called "SUCCESS". The system is schedulable if there exists at least one trace such that a location "SUCCESS" is reachable. Formally, we make the following assumption for the remaining of this chapter.

The system $\mathscr{S}$ is modeled by a parametric timed automaton $\mathscr{A}$ such that, for any valuation π, one can infer the schedulability of $\mathscr{S}[\pi]$ by looking at the set of locations of $\mathscr{A}[\pi]$. The system is schedulable if and only if at least one location "SUCCESS" is reachable.

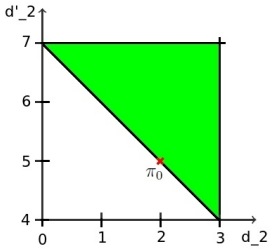
Under this assumption, *IM* can measure the robustness of the system around π_0. Indeed, since the trace set of $\mathscr{A}[\pi]$ is the same as for $\mathscr{A}[\pi_0]$ for any $\pi \models IM(\mathscr{A}, \pi_0)$, then the set of reachable locations is the same too. Hence the system $\mathscr{S}[\pi]$ is schedulable for any $\pi \models IM(\mathscr{A}, \pi_0)$ if and only if $\mathscr{S}[\pi_0]$ is schedulable. Note that, since we are interested in the preservation of *locations*, which is a weaker property than the equality of traces, we can use variants of the inverse method that preserve (at least) locations. This is in particular the case of the inverse method with merging 3.

4.2.2 Robustness Analysis Using the Inverse Method

Let us illustrate the application of *IM* to scheduling problems by analyzing the robustness of the preemptive jobshop example of [AM02] around the valuation $\pi_0 : \{d_2 = 2, d_2' = 5\}$, for the bound $\mu = 10$. We first consider a parametric version of $\mathscr{A}$ where d_2 and d_2' become parameters. We then apply *IM* to $\mathscr{A}$ and π_0; the resulting constraint K_0 is given in Figure 4.3(a), with its geometrical representation in Figure 4.3(b). From the correctness of *IM*, the trace set of $\mathscr{A}$ is always the same, for any point (d_2, d_2') of K_0. This trace set is depicted under the form of a graph in Figure 4.4 (recall that this graph representation is for sake of conciseness only; the trace set is a set of traces). Here, although many branches of the tree finish in a FAILURE location, there are also several branches that end in a SUCCESS location. These branches correspond to the schedules which are completed within $\mu = 10$ time units. The system is thus schedulable, for any

$$\begin{array}{rl} & d_2' < 7 \\ \wedge & d_2 < 3 \\ \wedge & d_2' + d_2 \geq 7 \end{array}$$

(a) Constraint K_0



(b) Geometrical representation

Figure 4.3: Application of *IM* to [AM02] with $\pi_0 : \{d_2 = 2, d_2' = 5\}$

point (d_2, d_2') of K_0. For example, we can increase d_2 from 2 to 3, or increase d_2' from 5 to 7 while keeping the completion time less than or equal to 10.

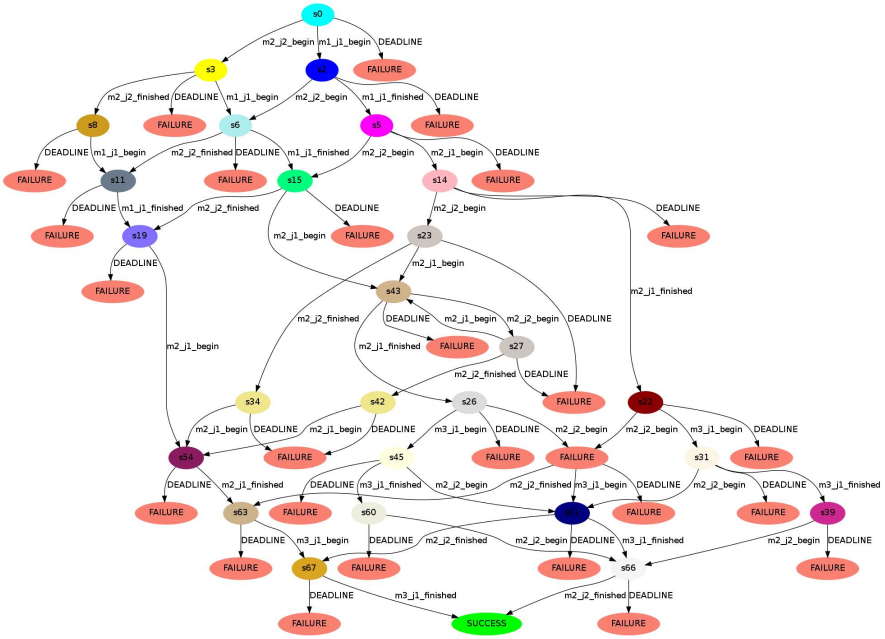


Figure 4.4: Trace set for the jobshop example

4.2.3 Schedulability Zone Synthesis

We now apply the behavioral cartography of Chapter 2.7 to solve the problem of synthesis of the schedulability zone (Problem 2) for the preemptive jobshop example of [AM02]. Let us consider a given rectangle V_0, say $[0, 11] \times [0, 11]$, and let us apply the *BC* method. We apply *IM* iteratively by letting π_0 equal to all the

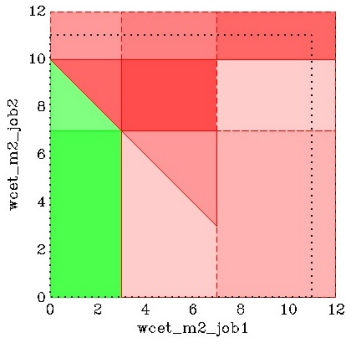


Figure 4.5: Schedulability zones (in green, the system is schedulable)

possible integer values of V_0. We thus synthesize different constraints K, which characterize different "behavior tiles". For any point of a tile, the behavior is uniform: either the system is schedulable (i.e., the set of feasible schedules is non empty) everywhere in the tile or nowhere in the tile. After 10 iterations, the rectangle V_0 (actually, the whole real-valued plan) is covered by the tiles generated successively. This is depicted in Figure 4.5. The green (resp. red) zone corresponds to the schedulable (resp. non-schedulable) zone.

4.3 Application to Scheduling Problems

In this section, we apply the approach of Section 4.2 to several scheduling problems from the literature, viz., the schedulability of jobs with deadlines (Section 4.3.1), a problem of synthesis of schedulability zone (Section 4.3.2), and an industrial case study designed by ASTRIUM (Section 4.3.3). All experiments have been performed using tool IMITATOR [AFKS12], that can also be applied to timed automata augmented with stopwatches.

4.3.1 Jobs with Deadlines

We consider here a system considered in [CPR08b, LPP+10] with a set of jobs $\{J_1,\ldots,J_n\}$. Each job J_i is periodic of period T_i (a fixed duration of time between two activation events), and an offset O_i for its first activation time. Once a job J_i has been activated, it executes for at most time C_i and has to terminate within the deadline D_i. The system is *schedulable* if each job J_i is completed before its relative deadline D_i[2]. We consider the case of two periodic jobs $\{J_1,J_2\}$ with $D_1=7,T_1=10,O_1=0,C_1=3,D_2=6,T_2=10,O_2=3,C_2=5$. We parameterize

[2]Actually, because of the periodicity of the system, we only have to be sure that it is schedulable within the least common multiple of the T_is, for $i=1\ldots n$.

C_1, C_2 and O_2. Applying *IM*, we find the constraint K_0 given in Figure 4.6(a). In [CPR08b], the authors use a CEGAR-based method to synthesize a constraint on the parameters, recalled in Figure 4.6(b), that guarantees that the system is schedulable. This latter constraint is incomparable with the constraint K_0.

$$
\begin{array}{ll}
 & 6 \geq C_2 \\
\wedge & 3 \geq C_1 \\
\wedge & 6C_1 > 17 \\
\wedge & 2C_1 + C_2 > 6 + O_2 \\
\wedge & 10 - C - 2 \geq O_2 \geq C_1
\end{array}
$$

(a) Constraint by *IM*

$$
\begin{array}{ll}
 & C_1 + C_2 < 6 + O_2 \\
\wedge & 6 < C_1 + C_2 < 10 \\
\wedge & C_2 < 10 - O_2 \\
\wedge & C_1 < 7 \\
\wedge & C_2 < 6
\end{array}
$$

(b) Constraint by [CPR08b]

Figure 4.6: Constraints synthesized for the [CPR08b, LPP+10] case study

4.3.2 Schedulability Zone Synthesis

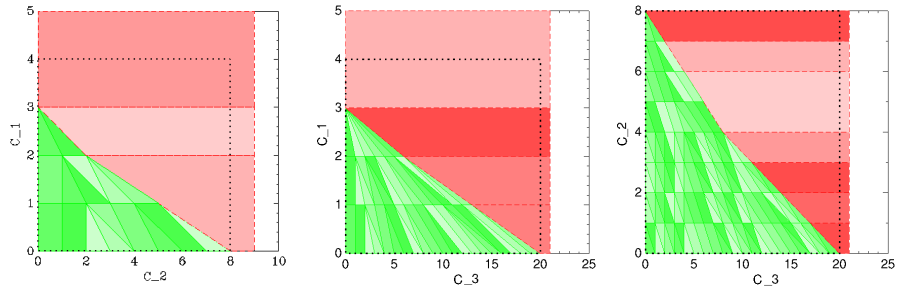
Let us apply the behavioral cartography method in order to determine zones of schedulability on an example with fixed priority ("Rate Monotonic") of [BB04a, Section III]. There are three periodic jobs J_1, J_2 and J_3 with periods of $T_1 = 3$, $T_2 = 8$ and $T_3 = 20$ and deadlines of $D_1 = 3$, $D_2 = 8$ and $D_3 = 20$. Our aim is to find a set of computation times C_i of each job τ_i ($1 \leq i \leq 3$) such that the system is schedulable, i.e., such that each job J_i is completed before T_i time units ($C_i \leq T_i$ for all $1 \leq i \leq 3$).

Let V_0 be the set of triples (C_1, C_2, C_3) ranging over $[0,3] \times [0,8] \times [0,20]$. Algorithm *BC* outputs a set of tiles, and it suffices to check one point per tile to determine the schedulability of the whole tile. The result for this example is given in Figure 4.7, using a discretization step of 0.2 on V_0. Since IMITATOR cannot output graphics in 3 dimensions, we project onto C_1 and C_2 (with $C_3 = 0$), onto C_1 and C_3 (with $C_2 = 0$), and onto C_2 and C_3 (with $C_1 = 0$), in Figures 4.7(a), 4.7(b) and 4.7(c), respectively (as it is done in [BB04a]). In each case, the green zone corresponds exactly to the schedulability region found using analytic methods in [BB04a, Figure 1(a)].

4.3.3 Next Generation Spacecraft Flight Control System

General Description

We describe here a prospective architecture for the flight control system of the next generation of spacecrafts designed by ASTRIUM Space Transportation. This work is part of a global project preparing the next generation of launcher



(a) C_1 and C_2 (with $C_3 = 0$) (b) C_1 and C_3 (with $C_2 = 0$) (c) C_2 and C_3 (with $C_1 = 0$)

Figure 4.7: Schedulability zones (in green the system is schedulable)

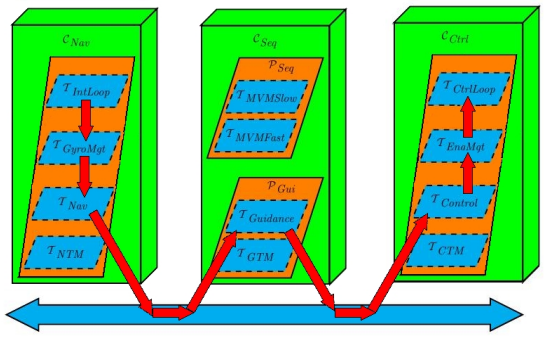


Figure 4.8: Architecture scheme

avionics architecture [MGS12]. In this design, the architecture is distributed on three processors (C_{Nav}, C_{Seq}, C_{Ctrl}) devoted to the treatment of information coming from the sensors, the computational analysis of the data, and the management of data to be sent to the actuators. The software running on each processor unit is organized into several *partitions*. Each partition contains itself several programs devoted to specific *tasks*. This is described in Figure 4.8: the outer boxes (in green) correspond to processors, the intermediate boxes (in orange) to partitions, and the inner boxes (in blue) to tasks. Each task τ is periodic, and characterized by a triple (O, C, T) of timings, where O corresponds to the offset, C to the execution time, and T to the period. Within a given partition, tasks are preemptible and scheduled according to a fixed priority scheduler, called "Rate Monotonic": the priority between two activated tasks is given to the task with the smaller period.

Tasks belonging to different partitions on a same processor are independent and can preempt each other. The preemption of one task of some partition by a task of another partition is called a *partition switch*. Partition switches are performed at predefined moments of time (i.e., are time triggered). An expected

Task	**Abbreviation**	(O,C,T)
Integration loop	*Intloop*	$(0,8,20)$
Gyroscope Management	*GyroMgt*	$(8,20,200)$
Navigation	*Navigation*	$(16,8,40)$
Navigation telemetry	*NTM*	$(2,10,50)$
Mission and Vehicle Management (slow)	*MVMSlow*	$(2,60,200)$
Mission and Vehicle Management (fast)	*MVMFast*	$(0,4,20)$
Guidance	*Guidance*	$(117,40,20000)$
Guidance telemetry	*GTM*	$(1,60,200)$
Control loop	*CtrLoop*	$(0,5,20)$
Engine management	*EngMgt*	$(15,12,50)$
Control	*Control*	$(1,15,100)$
Control telemetry	*CTM*	$(50,25,200)$

Table 4.1: Classical valuation of the parameters

output of the scheduling problem is the values of these moments (i.e., the *start time of activation* and *end time of activation* of each partition). There are thus *a priori* several sources of nondeterminism:

1. Inside a same processor, there are switches of partitions, and

2. the interleavings between the tasks processed by the different processors.

In addition, the system is organized into *jobs* (or "end-to-end flows"): each job J_i is described as a sequence of tasks $\{J_{i,1}, J_{i,2}, \ldots\}$. ($J_{i,j}$ cannot execute until its immediate predecessor $J_{i,j-1}$ completes.) A *deadline* D_i is associated with each job J_i: the last task of the job has to be completed before the deadline. The end-to-end flow is depicted in Figure 4.8 using the sequence of arrows (in red).

Note that, by comparison, the architecture of the flight control system running presently on the ASTRIUM Space Transportation spacecrafts is generally monoprocessor and mono-partition (see, e.g., [BDP10]).

Reference Parameter Valuation

A classical valuation π_0 of the triple (O,C,T) can for instance be the one given in Table 4.1, where O denotes the offset, C the execution time, and T the period. The job (end-to-end flow) J considered corresponds to the list (*IntLoop*, *GyroMgt*, *Navigation*, *Guidance*, *CTM*, *Control*, *EngMgt*, *CtrLoop*). The associated deadline is $D = 300$.

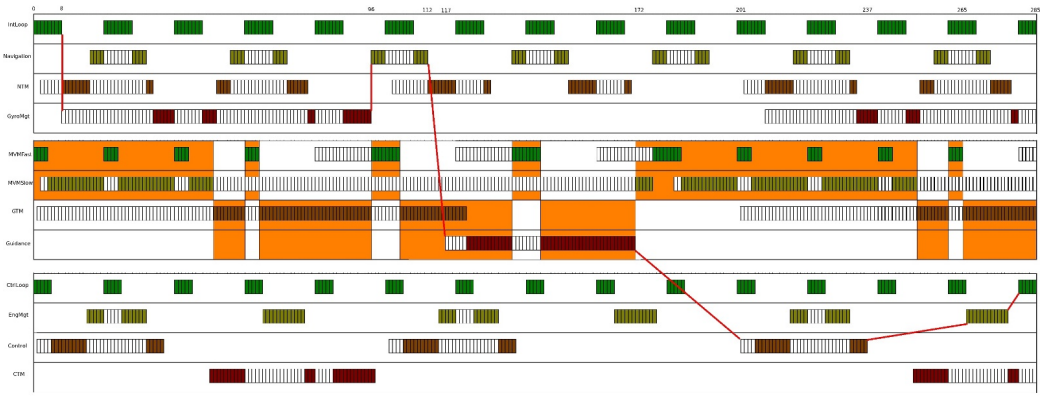


Figure 4.9: Chronogram of a schedule for J

Quantitative Robustness Analysis

We analyze the system from a quantitative robustness point of view following two steps. In a first step, we apply a standard reachability analysis to the system instantiated with π_0, and generate all the feasible schedules that satisfy the deadline D. This can be performed using most tools for the verification of (nonparametric) timed systems (IMITATOR actually also implements this feature). Among these schedules, we focus on a schedule that minimizes the number of partition switches. This schedule is depicted in Figure 4.9, under the form of a chronogram. One division of the time corresponds to 1 time unit, and the job is completed after 285 time units. The upper level of large, plain rectangles (in orange) indicates the running of the MVMFast-MVMSlow partition while the lower level indicates the running of the GTM-Guidance partition on processor $\mathscr{C}_{Seq}$. In the ASTRIUM project design, the partition switches are time-triggered: the switches are performed at predefined moments. For example, the admissible schedule depicted in Figure 4.9 can be seen as a way of programming the partition switches on the second processor $\mathscr{C}_{Seq}$ (corresponding to border between contiguous orange rectangles) at times: $51, 60, 64, 96, 104, 136, 144, 171, 251, 260, 264$.

In a second step, we apply this method to analyze the robustness of the execution times and offset times while keeping the above time-triggered sequence of partition switchings. This is done by imposing the values of time of partition switches, as specified above, and parameterizing all the execution times Cs and offsets Os corresponding to tasks *MVMFast*, *MVMSlow*, *GTM* and *Guidance*. The method then outputs the following constraint K_0:

$$
\begin{array}{lll}
 & 4 \geq C_{MVMFast} > 1 & \wedge \quad 8C_{MVMFast} + C_{MVMSlow} > 71 \\
\wedge & C_{GTM} > 57 & \wedge \quad 100 \geq C_{Guidance} + C_{GTM} > 89 \\
\wedge & 120 > O_{Guidance} \geq 55 + C_{GTM} & \wedge \quad O_{Guidance} > C_{MVMFast} + C_{MVMSlow} \\
\wedge & C_{MVMFast} > O_{MVMSlow} > O_{GTM} > 0 & \wedge \quad O_{MVMFast} = 0
\end{array}
$$

For any tuple of values satisfying the constraint K_0, the time-triggered schedule of Figure 4.9 is still valid.

In more classical approaches, tools only compute a solution of the scheduling problem. The engineers using this automatically computed result may then completely lose the feeling of their system; in particular, they cannot quantify the robustness of the design with respect to, for instance, small variations of worst case execution times or delay deadlines. In contrast, the constraint K_0 indicates clearly to the designer some degrees of freedom, allowing a better mastering of the margin policy.

4.4 A Comparison with Analytic Method

4.4.1 Analytic Method

In this section we present a novel method for parametric analysis of distributed system. The method extends the sensitivity analysis by Bini *et al.* [SLS98, Bin04] to include jitter and deadline parameters.

In Sections 4.4.1 and 4.4.2, we only consider the scheduling of independent periodic tasks in a single processor. Then, in Section 4.4.2, we extend the schedulability analysis to distributed systems.

Preemptive Tasks with Constrained Deadlines

There are many ways to test the schedulability of a set of real-time periodic tasks scheduled by fixed priority on a single processor. In the following, we will use the test proposed by Seto *et al.* [SLS98] because it is amenable to parametric analysis of computation times, jitters and deadlines.

The original theorem was formulated for tasks with deadlines equal to periods. For the moment, we generalise it to tasks with constrained deadlines (i.e. $D_i \leq T$), while in Section 4.4.2 we deal with unconstrained deadlines, jitter and non-preemptive scheduling.

Definition 30. *The set of* scheduling points $\mathbb{P}^{i-1}(t)$ *for a task* τ_i *is the set of all vectors corresponding to multiples of the period of any task* τ_j *with priority higher than* τ_i*, until the maximum possible value of the deadline. It can be computed as follows. Let* $\eta_j(t) = \left\lceil \frac{t}{T_j} \right\rceil$*, and let* $\eta^{i-1}(t)$ *be the corresponding vector of*

$i-1$ *elements with* $j = 0,\ldots,i-1$. *Then:*

$$\mathbb{P}^{i-1}(t) = \{\eta^{i-1}(t)\} \cup \{\eta^{i-1}(kT_h) \mid 0 < kT_h < t, h < i\} \tag{4.1}$$

Theorem 8 ([SLS98])**.** *Consider a system of periodic tasks* $\{\tau_1,\ldots,\tau_n\}$ *with constrained deadlines and zero jitter, executed on a single processor by a fixed priority preemptive scheduler. Assume all tasks are ordered in decreasing order of priorities, with* τ_1 *being the highest priority task.*

Task τ_i *is schedulable if and only if:*

$$\exists \mathbf{n} \in \mathbb{P}^{i-1}(D_i) \begin{cases} C_i + \sum_{j=1}^{i-1} n_j C_j \le n_k T_k \quad \forall k = 1,\ldots,i-1 \\ C_i + \sum_{j=1}^{i-1} n_j C_j \le R_i \\ R_i \le D_i \end{cases} \tag{4.2}$$

where $\mathbf{n}$ *is a vector of* $i-1$ *integers, and* $\mathbb{P}^{i-1}(D_i)$ *is the set of scheduling points.*

Notice that, with respect to the original formulation, we have separated the case of $k = i$ from the rest of the inequalities and we introduced variable R_i.

The theorem allows us to only consider sets of linear inequalities, because the non-linearity has been encoded in the variables n_j. Each vector $\mathbf{n}$ defines a convex region (maybe empty) with variables $C_1,\ldots,C_i$ and $R_1,\ldots,R_i$. The "*exists*" quantifier means that the region for each task τ_i is the union of convex regions, hence it may be non-convex. Since we have to check the schedulability of all tasks, we must intersect all such regions to obtain the final region of schedulable parameters. The resulting system is a disjunction of sets of conjunctions of inequalities. Geometrically, this corresponds to a non-convex polyhedron in the space of the variables C and R of tasks.

It is worth to note that, using this formulation, we can compute the response time of a task by simply minimising the corresponding variable R_i under the constraints of Equation (4.2). As an example, consider the following task set (the same as in [BB04b]): $\tau_1 = (C = 1, T = 3), \tau_2 = (C = 2, T = 8), \tau_3 = (C = 4, T = 20)$, in decreasing order of priority, to be scheduled by preemptive fixed priority scheduling on a single processor.

We consider the response time R_3 as a parameter and set up the system of inequalities according to Equation (4.2). After reduction of the non-useful constraints, we obtain $12 \le R_3 \le 20$. Therefore, the response time is $R_3 = 12$, which is the same that can be obtained by classical response time analysis.

4.4.2 Extensions to the Model

We now extend Seto's test to unconstrained deadlines and variable jitters, and non-preemptive scheduling. Non-preemptive scheduling can be modelled by considering an initial *blocking time*, due to the fact that a task cannot preempt lower-priority executing tasks.

The worst case response time for a non preemptive task τ_i can be found in its longest i-level active period [BLV07]. An i-level active period L_i is an interval $[a,b)$ such that the amount of processing that needs to be performed due to jobs with priority higher than or equal to τ_i (including τ_i itself) is larger than 0 for all $t \in (a,b)$, and equal to 0 at instants a and b. The longest L_i can be found by computing the lowest fixed point of a recursive function. Notice that, by considering non-preemption and tasks with deadline greater than periods, the worst-case response time may be found in any instance of the active period, not necessarily in the first one (as with the classical model of constrained deadline preemptive tasks).

Unfortunately, the longest busy period cannot be computed when tasks have parametric worst-case computation times. However, under the assumption that there is at least an idle-time in the hyperperiod (i.e. its utilisation is strictly less than 100%) a sufficient feasibility test can be derived by computing the worst-case response time for every instance of the task set in the hyperperiod H_n. Therefore, we can extend our model as follows.

Theorem 9. *A non preemptive task τ_i is schedulable if $\forall h = 1,\ldots,\frac{H_n}{T_i}$, $\exists \mathbf{n} \in \mathbb{P}^{i-1}((h-1)T_i + D_i)$ such that*

- $B_i + (h-1)C_i + \sum_{j=1}^{i-1} n_j C_j \leq n_l T_l - J_l \quad \forall l = 1,\ldots,i-1;$
- $B_i + (h-1)C_i + \sum_{j=1}^{i-1} n_j C_j \leq (h-1)T_i + R_i - C_i - J_i;$
- $R_i \leq D_i$ *and* $B_i \leq C_j - 1$ *for all* $j > i$.

Proof. See [SSL+13b]. □

Term B_i is an additional internal variable used to model the blocking time that a task suffers from lower priority tasks. It is possible to avoid the introduction of this additional variable by substituting it in the inequalities with a simple Fourier-Motzkin elimination.

Notice that the introduction of unconstrained deadlines adds a great amount of complexity to the problem. In particular, the number of non-convex regions to intersect is now $\mathcal{O}(\sum_{i=1}^{n} \frac{H_n}{T_i})$, which is dominated by $\mathcal{O}(nH_n)$. So, the

proposed problem representation is pseudo-polynomial in the size of the hyperperiod. However, in real applications, we expect the periods to have "nice" relationships: for example, in many cases engineers choose periods that are multiples of each others. Therefore, we expect the set of inequalities to have manageable size for realistic problems.

Distributed Systems

Until now, we have considered the parametric analysis of independent tasks on single processor systems, with computation times, response times, blocking times and jitters as free variables.

One key observation is that a precedence constraint between two consecutive tasks τ_i and τ_{i+1} in the same pipeline can be expressed as $R_i \leq J_{i+1}$. This relationship derives directly from the definition of response time and jitter in Section 4.1.3. Using this elementary property, we can now build the parametric space for a distributed system as follows.

1. For each processor and network, we build the constraint system of Theorem 9. Notice that the set of constraints for the individual single processor systems are independent of each other (because they are constraints on different tasks).

2. For each pipeline $\mathscr{P}^a$:

   - two successive tasks τ_i^a and τ_{i+1}^a must fulfil the constraint $R_i^a \leq J_{i+1}^a$;
   - for the initial task we impose $J_1^a = 0$.

Such pipeline constraints must intersect the combined system to produce the final system of constraints. However, simply adding the above precedence constraints can lead to pessimistic solutions. In fact, if two tasks from the same pipeline are assigned to the same processor, the interference they may cause on each other and on the other tasks may be limited.

Suppose τ_i^a and τ_j^a are allocated to the same processor and $q_i^a > q_j^a$. Then, τ_i^a can at most interfere with the execution of a job from τ_j^a a number of times equal to $\xi = \left\lceil \frac{\max\{0, D_{e2e}^a - T^a\}}{T^a} \right\rceil$. So, we impose that $\forall \mathbf{n} \in \mathbb{P}^{j-1}$, $n_i \leq \xi$.

The analytic method proposed in this section has been implemented in a software tool, called RTSCAN, which is based on the PPL (Parma Polyhedra Library) [BHZ08], a library specifically designed and optimised to represent and operate on polyhedra. The library efficiently operates on rational numbers with arbitrary precision: therefore, in this work we make the assumption that all

variables (computations times, response times and jitter) are defined in the domain of rationals (rather than reals).

We observed that the complexity of the methodology for generating the parameter space strongly depends on the number of free parameters considered in the analysis. Therefore, as a preliminary step, the tool requires the user to select a subset of the computation times on which the analysis will be performed, whereas the other parameters will be assigned fixed values. During construction of the polyhedron we have to keep R_i, J_i and B_i for each task as variables. Therefore, the number of variables to be managed is $\mathrm{nV} = 4 \cdot N + F$, where N is the number of tasks and F is the number of variables to analyse. At the end, we can eliminate the R_i, J_i and B_i variables, hence the final space consists of F dimensions. An evaluation of this tool and of the run-time complexity of the analysis will be presented in Section 4.4.3.

The analytic method described so far is not exact. In fact, when dealing with pipelines in a distributed system we may sometimes overestimate the interference of higher priority-tasks on lower priority ones. For this reason, we now present an exact parametric analysis based on PSA and model checking.

4.4.3 Comparison

In this section we evaluate the effectiveness and the running time of the two proposed tools on two case studies. As a baseline comparison, we choose to also run the same kind of analysis on the same case studies using MAST.

In order to simplify the visualisation of the results, for each test case we present the 2D region generated for two parameters only. However, all three methods are general and can be applied to any number of parameters. In Section 4.4.3 we will present the execution times of the three tools on the test-cases.

MAST [GHGGPGDM01] is a software tool implemented and maintained by the CTR group at the *Universidad de Cantabria* that allows to perform schedulability analysis for distributed real-time systems. It provides the user with several different kinds of analysis. For our purposes, we have selected the "Offset Based analysis" [PGH98], an improvement over classical holistic analysis that takes into account some of the relationships between tasks belonging to the same pipeline.

Test Case 1

The first test case (TC1) has been adapted from [PGH98] (we reduced the computation times of some tasks to position the system in a more interesting schedulability region). It consists of three simple periodic tasks and one pipeline, running on two processors (p_1 and p_3), connected by a CAN bus (p_2).

Pipeline/Task	T	D_{e2e}	Tasks	C	q	p
τ_1	20	20	-	free	9	1
P^1	150	150	τ_1^1	free	3	1
			τ_2^1	10	9	2
			τ_3^1	8	5	3
			τ_4^1	15	2	2
			τ_5^1	25	2	1
τ_2	30	30	-	6	9	3
τ_3	200	200	-	40	2	3

Figure 4.10: TC1 – all numbers in "ticks"

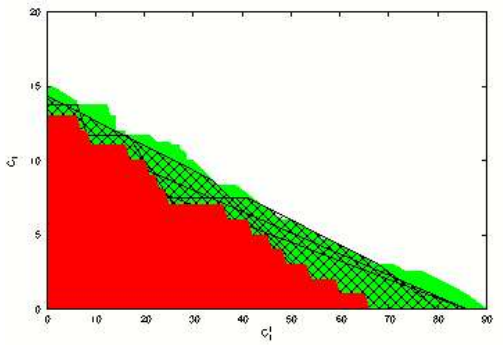


Figure 4.11: TC1: Schedulability regions produced by RTSCAN (hatched), MAST (red, below), and IMITATOR (green, above)

The parameters are listed in Figure 4.10. The pipeline models a remote procedure call from processor 1 to processor 3. All tasks have deadlines equal to periods, and also the pipeline has end-to-end deadline equal to its period. Only two messages are sent on the network, and according to our optimisation rule for building parametric space, if the pipeline is schedulable, they cannot interfere with each other. We performed parametric schedulability analysis with respect to C_1 and C_1^1.

The resulting regions of schedulability from the three tools are reported in Figure 4.11. In this particular test, RTSCAN dominates MAST. After some debugging, we discovered that the analysis algorithm currently implemented in MAST does not consider the fact that the two messages τ_2^1 and τ_4^1 cannot interfere with each other, and instead considers a non-null blocking time on the network.

As expected, the region computed by IMITATOR dominates the other two tools. This means that there is much space for improvement in the analysis even for such simple systems.[3]

Test Case 2

The second test case is taken from [WTVL06]. It consists of two pipelines on 3 processors (with id 1, 3 and 4) and one network (with id 2). We actually consider two versions of this test case: in the first version (a) pipeline P^1 is periodic with period 200 ms and end-to-end deadline equal to the period. In the second

[3] By zooming in the figure, it looks like in some very small areas, the region produced by RTSCAN goes over the region produced by IMITATOR. However, remember that both tools only deal with integer numbers; that small region does not contain any integer point.

Pipeline	T	D_{e2e}	Tasks	C	q	p
P^1	200 (30)	200	τ_1^1	4,546	10	1
			τ_2^1	445	10	2
			τ_3^1	9,091	10	4
			τ_4^1	445	9	2
			τ_5^1	free	9	1
P^2	3,000	1,000	τ_1^2	free	9	4
			τ_2^2	889	8	2
			τ_3^2	44,248	10	3
			τ_4^2	889	7	2
			τ_5^2	22,728	8	1

Figure 4.12: Test case 2: periods and deadlines are in milliseconds, computation times in micro-seconds.

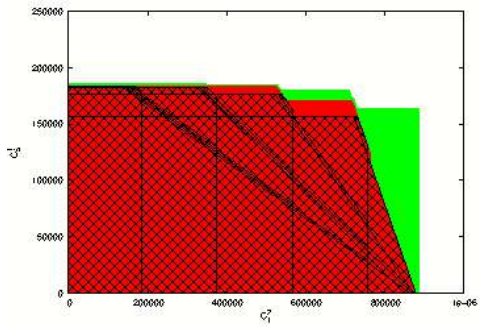


Figure 4.13: Schedulability regions for test case 2a, produced by RTSCAN (hatched), MAST (red), and IMITATOR (green)

version (b), the period of the first pipeline is reduced to 30 ms (as in the original specification in [WTVL06]). The full set of parameters is reported in Table 4.12, where all values are expressed in microseconds. We perform parametric analysis on C_5^1 and C_1^2.

For version (a) we run all tools and we report the regions of schedulability in Figure 4.13. Once again IMITATOR dominates the other two. Also, MAST dominated RTSCAN. The reason is due to the offset-based analysis methodology used in MAST, which reduces the interference on one task from other tasks belonging to the same pipeline.

For version (b) we run only RTSCAN and MAST, because in the current version we only model constrained deadline systems with IMITATOR. The results for version (b) are reported in Figure 4.14. In this case, MAST dominates RTSCAN. Again, this is due to the fact that MAST implements the offset-based analysis.

Execution Times

Before looking at the execution times of the three tools in the three different test cases, it is worth to discuss some details about their implementation.

IMITATOR produces a disjunction of convex regions. However, these regions are typically small and disjoints. Moreover, to produce a region, IMITATOR needs to start from a candidate point on which to call *IM*, and then move to close-by regions. One key factor here is how this search is performed. Currently, IMITATOR searches for a candidate point in the neighbourhood of the current

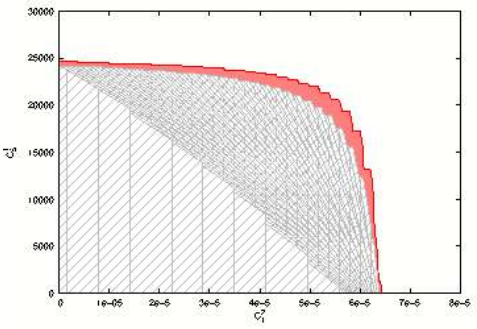


Figure 4.14: Schedulability regions for test case 2b, produced by RTSCAN (grey, below) and MAST (red, above)

Test Case	RTSCAN	MAST	IMITATOR
1	0.27s	7 s	19min42
2a	0.47s	40min13	2h08
2b	1min11	33min19	–

Figure 4.15: Execution times of the tools

region. This is a very general strategy that works for any kind of PSA. However, the particular structure of schedulability problems would probably require an ad-hoc exploration algorithm.

MAST can perform sensitivity analysis on one parameter (called *slack computation* in the tool), using binary search on a possible interval of values. Therefore, to run the experiments, we performed a cycle on all values of one parameter (with a predefined step) and we asked MAST to compute the interval of feasible values for the other parameter.

All experiments have been performed on an Intel Core I7 quad-core processor (800 MHz per processor) with 8 GiB of RAM. The execution times of the tools in the three test cases are reported in Figure 4.15. RTSCAN is the fastest method in all test-cases. In test case 2b, the execution time of RTSCAN is much larger than the one obtained from test case 2a. This is due to the fact that in test case 2b one pipeline has end-to-end deadline greater than the period, and therefore RTSCAN needs to compute many more inequalities (for all points in the hyperperiod). Finally, IMITATOR is the slowest of the three and does not scale well with the size of the problem. We observed that the tool spends a few seconds for computing the schedulability region around each point. However, the regions are quite small, and there are many of them: for example, in test case 2a IMITATOR analysed 257 regions. Also, the tool spends a large amount of time in searching for neighbourhood points. We believe that some improvement in the computation time of IMITATOR can be achieved by coming up with a different exploration strategy more specialised to our problem.

We also evaluated the scalability of RTSCAN with respect to the number of parameters. To do this, we run the tool on test case 2b with a varying number of parameters. The computation time went from 1min11 for $F = 2$ parameters, up to 20min15 for the case of $F = 6$. With $F = 6$, the memory used by our program

took a peak utilisation of 7.2 GiB, close to the memory limit of our PC. However, we believe that 6 parameters are sufficient for many practical engineering uses.

4.5 Discussion

As shown on different case studies of the literature, this procedure provides the designer with a uniform method for evaluating quantitatively the robustness of scheduling solutions. Furthermore, as exemplified on an industrial case study, this approach is able to manage a large scope of industrial problems in the domain of critical embedded software. Compared to classical approaches, it automates a boring error-prone manual activity and it formalizes the margins of evolutions of the system (margins which are generally only estimated without formally insurance of validity).

However, in spite of first promising successes, this approach meets a combinatory explosion problem when faced with even more sophisticated space systems designs that integrate more partitioning and distributed computing. Adaptations of the method in order to tackle such highly distributed-computing architectures should be performed.

4.6 Related Work

The use of models such as parametric timed automata and parametric time Petri nets for solving scheduling problems has received attention in the past few years. The approach most related to the method described in this chapter is [CPR08b, LPP+10], where the authors infer parametric constraints guaranteeing the feasibility of a schedule, using parametric timed automata with stopwatches. The main difference here relies in the choice here of the inverse method, rather than a CEGAR-based method. First results obtained on the same case studies are incomparable (although similar in form), which seems to indicate that the two methods are complementary. The problem of finding the schedulability region was attacked in analytic terms in [BB04a]; the size of the examples considered in this chapter is rather modest compared to those treated using such analytic methods. However, in many schedulability problems, no analytic solution exists (see, e.g., [SGL97]), and exhaustive simulation is exponential in the number of jobs. In such cases, symbolic methods as the inverse method and those of [CPR08b, LPP+10] are useful to treat critical real-life examples of small or medium size, as exemplified here in Section 4.3.3.

Many research papers have already addressed the problem of parametric schedulability analysis, especially on single processor systems. Bini and But-

tazzo [Bin04] proposed an analysis of fixed priority single processor systems, which is used as a basis for this paper.

Parameter sensitivity can be also be carried out by repeatedly applying classical schedulability tests, like the holistic analysis [PGH98]. One example of this approach is used in the MAST tool [GHGPD01], in which it is possible to compute the *slack* (i.e. the percentage of variation) with respect to one parameter for single processor and for distributed systems by applying binary search in that parameter space [PGH98].

A similar approach is followed by the SymTA/S tool [HHJ+05], which is based on the *event-stream* model [RE02]. Another interesting approach is the Modular Performance Analysis (MPA) [WTVL06], which is based on Real-Time Calculus. In both cases, the analysis is compositional, therefore less complex than the holistic analysis. Nevertheless, these approaches are not fully parametric, in the sense that it is necessary to repeat the analysis for every combination of parameter values in order to obtain the schedulability region.

Model checking of *parametric timed automata* (PTA) or *parametric stopwatch automata* (PSA) can be used for parametric schedulability analysis [CPR08a]. In particular, thanks to generality of the PTA and PSA modelling language, it is possible to model a larger class of constraints, and perform parametric analysis on many different variables, for example task offsets. This approach has been recently extended to distributed real-time systems [LPPR13].

Also grounded on PTA and PSA is the Inverse Method [AS13], applied in particular to schedulability analysis [FLMS12]. This method is very general because it permits to perform analysis on any system parameter. However, this generality may be paid in terms of complexity of the analysis.

In this paper, we aim at performing fully parametric analysis of real-time distributed systems. We first present extensions of the methods proposed in [Bin04] to the case of distributed real-time systems. We also present a model of a distributed real-time systems using PSA, and compare the two approaches against classical analysis in MAST.

Chapter 5

Conclusion and Perspectives

Multiprocessor and multicores architectures are widely accepted for personal use. These systems allow for faster, cheaper and less energy consuming computations. However, due to the difficulty to certify them, industrial use for critical systems is marginal. Formal methods can be useful but often suffer from the state space explosion.

The Inverse Method was designed to work on PTA to verify asynchronous circuits, communication protocols and so on. We have shown how to extend it to stopwatches instead of clocks in order to verify scheduling with preemption. To overcome the state space explosion, we have extended a state merging technique on TA to PTA by using convexity properties. We have shown that some interesting properties can still be verified when using merging.

Our approach has then been applied to several case studies from the literature and to an industrial case study. We have designed a scheduler for the Astrium case study and we have provided a constraint on the parameters so that this scheduler is correct no matter what parameter valuation is chosen inside this constraint. This allow the design team some freedom to modify the parameters valuation without having to go through a costly testing phase. The first results are very promising and we think that our method can be useful and may scale up by tuning our approach to this type of case study by using theorems from the schedulability community or using model reduction.

The Behavioral Cartography is of interest when solving the schedulability problem because it is exact whereas most traditional techniques are under-approximation. Moreover, in our approach, periods, deadlines, WCETs can be parametrized all together whereas mos technique in the literature only focus on the parametrization of periods or deadlines or WCETs.

To scale up even more, an interesting approach would be to use model reduction that can still be modeled as PTA. Verification on these reduced models would be faster while still allowing the verification of properties of interest.

Part II

Controllability of Sampled Switched Systems

Chapter 6

Introduction

In recent years, there has been an increasing interest in optimizing the energy use in electricity generation and transportation. In particular, much effort has been devoted to the improvement of robust and flexible control techniques of power converters in order to increase reliability and safety of operation. Due to their practical feasibility to achieve a high performance as well as natural digital implementation in signal processors, *switched controllers* are the most commonly type of controller that has been applied to power converters (see [Lib03, SG05, CPPMT09]). Systems equipped with switched controllers are constituted of two parts: first, a family of continuous subsystems or *modes*; second, a *switching signal* that orchestrates the selection of these modes. The switching signal can be state dependent and/or time dependent.

With respect to classical systems, an interest of switching controllers stems from the existence of systems that cannot be asymptotically stabilized by a single continuous feedback control law [Bro83]. However, with switched systems, the steady-state operating condition is typically a *periodic solution* or *limit cycle*, not an equilibrium point. The relevant stability notion is *asymptotic orbital stability* or *practical stability*, which studies the conditions under which the system state evolves within certain subsets of the state space [LL61]. The problem of stabilization of switched dynamical systems is thus much more difficult in general than in classical control theory. In particular, instability phenomena can occur even when all the modes, taken separately, are stable.

Although the general control theory of switched systems is very difficult, special cases of these problems arise frequently in restricted contexts associated with control design, and may be simpler to solve specifically. It is thus suggested in [LM99] to stay in close contact with particular applications of switched systems. We have thus focused on switching signals that operates with a fixed switched period denoted by τ. These signals are very common because of their ease of implementation. Also a fixed period operation avoids

potentially troublesome harmonic side-effects that may arise with varying frequency operation (see [GPM08]). There are two types of periodic switched controllers: *state-independent* controllers that apply cyclically the same sequence of modes that has been computed off-line, or *state-dependent* controllers that select modes dynamically according to the regions of the states at the switching instants. Furthermore, the dynamics of each subsystem obeying to Ohm's electrical laws, are governed by *affine* differential equations. These systems can thus be viewed as special cases of *hybrid systems* (see [Hen96]) combining affine continuous dynamics and discrete transitions happen at instants that are integer multiples of τ. Such a subclass has been recently studied by many researchers such as Antoine Girard, Giordano Pola, and Paulo Tabuada, (see, e.g., [Tab09]). These systems are called "time-triggered sampled version of switched systems". We call them here simply $\mathscr{S}^2$-systems (for Sampled Switched Systems).

In classical control theory, one makes use of Lyapunov theory in order to analyze and stabilize controlled systems. Roughly speaking, Lyapunov functions are energy functions characterizing the state of the system which decrease until they reach a 0-level, which corresponds to a level where the system is stable. These Lyapunov functions can be extended in the framework of switched systems under the form of so-called "multiple Lyapunov functions" or "common Lyapunov functions", and play a central role in, e.g., [Tab09]. However, there is no general method for finding appropriate Lyapunov functions, and we have preferred in our approach to avoid to use them. Instead, the theoretical tool that we have mainly used is based on the notion of "(controlled) invariant" [Bla99]. Note however that the two concepts of invariants and Lyapunov functions are closely related, and one can show (at least in the classical context) that the level sets of Lyapunov functions correspond to the boundaries of invariant sets, and that the converse holds.

We focus on the restricted class of sampled switched systems, and on methods for controlling them using invariants. We will exploit the construction of invariants in order to synthesize two main classes of controllers: *safety* controllers and *stability* controllers. Safety controllers aim at protecting the system from undesirable states, while stability controllers aim at driving the system to a steady-state operating condition. In order to synthesize safety controllers we describe *indirect* methods working on an abstract discrete level, and *direct* methods working on the continuous state space level. These methods adopt a classical *backward* computation of the reachable states. In order to synthesize stability controllers, we describe a method of state space *decomposition* that allows to construct limit cyclic trajectories by iterated forward computation of the reachable states.

The control strategies synthesized by this method have been numerically

simulated, and also successfully experimented on physical prototypes built by SATIE Laboratory (Laboratoire des Systèmes et Applications des Technologies de l'Information et de l'Énergie), École Normale Supérieure de Cachan (ENS Cachan), France.

Organization of this Part

This part is structured as follows. In Chapter 7, we formally define the model of $\mathscr{S}^2$-systems, and explain in Section 7.5, how to synthesize safety controllers for $\mathscr{S}^2$-systems. In Chapter 8, we explain how to synthesize stability controllers for $\mathscr{S}^2$-systems using an original procedure of state space decomposition. In Section 8.6, it is explained how to extend the procedure in order to synthesize robust safety controllers and reachability controllers, and suggest to use it for sensitivity analysis. We show in Chapter 9 how to apply the procedure for controlling an important application of power electronics. The main results with some perspectives are reviewed in Chapter 10. Notes citing sources and related works are given at the end of each chapter.

Chapter 7

Control Theory: Basic Concepts

This chapter presents basic concepts of control theory, which will be used in the rest of this part.

Outline of the chapter

In Section 7.1, we present the classical *control/plant* model. In Section 7.2, we explain why the introduction of digital sensors and actuators in systems have fundamentally modified the issue of controlled stability. Finally we recall the model of *switched* systems, and explain their advantages compared with classical systems (Section 7.2.3). We then explain in Section 7.3 how the notion of *invariant sets* can be used for proving safety and stability properties of controlled systems. We give the formal model (Section 7.4.1) of $\mathscr{S}^2$-systems together with illustrative examples (Section 7.4.2). We also explain how to represent efficiently sets of states using the notion of "zonotope" (Section 7.4.3).

7.1 Model of Control Systems

A control system is classically decomposed into a controlled part, called *plant*, and a *controller*. The plant is classically described as a dynamic time-invariant, possibly uncertain, system governed by equations of the form:

$$\begin{cases} \dot{x}(t) = f(x(t), u(t), w(t)) & (1) \\ y(t) = g(x(t)) & (2) \end{cases}$$

where $x(t) \in \mathbb{R}^n$ is the *system state*, $u(t) \in \mathbb{R}^m$ is the *control input*, $y(t) \in \mathbb{R}^p$ is the *output*, $w(t) \in \mathscr{W} \subset \mathbb{R}^q$ is a *disturbance* (or external input), and $\mathscr{W}$ is an assigned compact set. We will refer to $\mathbb{R}^n$ as the *state space* of the system. The classical theory of control focuses on *feedback control*: the controller is fed with state signal $x(t)$ coming from the plant, and issues a control input $u(t)$ to the plant. A typical layout of a feedback control system is shown in Figure 7.1. Under classical conditions (continuity for u and w, and Lipschitz property for

f), the system admits a unique solution $x(t)$ on $\mathbb{R}_{\geq 0}$. The equations (1-2) are often simplified by disregarding $w(t)$, and assuming that $y = x$.

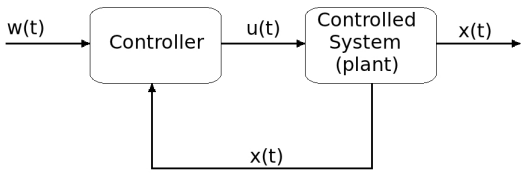


Figure 7.1: Control/plant model

An important subclass is the *linear time-invariant* (LTI) framework, for which (1-2) becomes:

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) + Ew(t) \\ y(t) = Cx(t) \end{cases}$$

for matrices A, B, C, E of appropriate size with constant coefficients. A *discrete-time LTI* system is a system governed by an equation of the form: $x(t+1) = Ax(t) + Bu(t) + Ew(t)$. When a system is governed by an equation of the form $\dot{x}(t) = Ax(t)$ where A is a matrix whose eigenvalues have negative real parts, the origin is a *stable equilibrium* point to which the system converges from any initial point of $\mathbb{R}^n$. Given a plant governed by an equation of the form $\dot{x}(t) = Ax(t) + Bu(t)$ with $(A, B) \in \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times m}$, a typical problem of linear control theory is to find a stabilizing controller governed by an equation of the form $u(t) = Kx(t)$ with $K \in \mathbb{R}^{m\times n}$. This essentially amounts to find coefficients values of K that make the real parts of the eigenvalues of $A + BK$ negative.

7.2 Digital Control Systems

7.2.1 Digitization

With the emergence of digital computers, a control system has to handle data that come from the periodic sampling of signals. In such a context, a control system is said to be *sampled-data* or *digital control system* . There, a system described by differential equations (which involve continuous-valued variables that depend on continuous time) is controlled by a discrete-time controller described by difference equations, which involve continuous-valued variables that depend on discrete time. As explained in [AK02], a digital control system, can be divided into three parts, the plant, interface, and controller as shown in Figure 7.2.

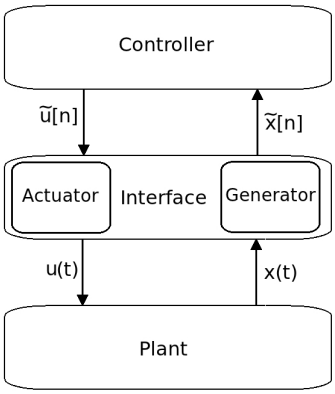


Figure 7.2: Digital control/plant model (from [AK02])

The system to be controlled (*plant*) is modeled as a time-invariant continuous-time system governed by equations (1-2) where, for the sake of simplicity, we disregard disturbance and assume that the output function is the identity (i.e., we have $\dot{x} = f(x,u)$ and $y = x$).

The *controller* is a discrete event system modeled as a deterministic automaton. The action of the controller can be described by equations of the form:

$$\begin{cases} \tilde{s}[n] = \delta(\tilde{s}[n-1], \tilde{x}[n]) \\ \tilde{u}[n] = \phi(\tilde{s}[n]) \end{cases}$$

where δ is the state transition function of the controller, and ϕ is the output function of the controller. Tildes are used to indicate that the particular signal is made up of symbols. The index n is here analogous to a time index in that it specifies the order of the symbols in the sequence. An argument in brackets, e.g., $\tilde{x}[n]$, represents the nth symbol from a set. The input signal $\tilde{x}$ and output signal $\tilde{u}$ associated with the controller are sequence of symbols, rather than continuous-time signals. Notice that there are *no delay* in the controller: the state transition, from $\tilde{s}[n-1]$ to $\tilde{s}[n]$, and the controller symbol, $\tilde{u}[n]$, occurs immediately when the plant symbol $\tilde{x}[n]$ occurs.

The controller and plant cannot communicate directly because each utilizes different types of signals. Thus, an *interface* is required that can convert continuous-time signals to sequences of symbols and vice versa. The interface consists of a memoryless map γ called *actuator*, and a memoryless map α called *generator*. The actuator converts a controller symbol $\tilde{u}[n]$ to a constant plant input of the form $u(t) = \gamma(\tilde{u}[n])$. Since the plant input, u, can only take on certain constant values, where each value is associated with a particular controller symbol, the plant input signal $u(t)$ is *piecewise constant*, and may change only when a controller symbol occurs. Such a piecewise continuous command

signal issued by the actuator is illustrated in Figure 7.3. The generator is a func-

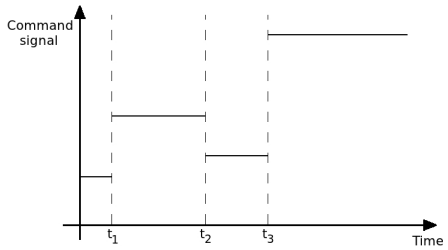


Figure 7.3: Staircase command signal $u(t)$ issued by the actuator as it receives controller symbols $\tilde{u}[1], \tilde{u}[2], \ldots$ at time t_1, t_2, ... (from [AK02])

tion α which maps the real-valued state vector $x(t)$ of the plant into a plant symbol of the form $\tilde{x}[n] = \alpha(x(t))$. Note that $\tilde{x}$ does not change continuously, but only when a *plant event* occurs. There are two different models of plant event: in the *state-triggered* model, a plant event occurs when the plant state x crosses the boundary of two pre-defined state regions; in the *time-triggered* model, a plant event occurs periodically when the signal $\tilde{x}$ issued by the generator corresponds to a *periodic sampling* of the plant output x, as illustrated in Figure 7.4.

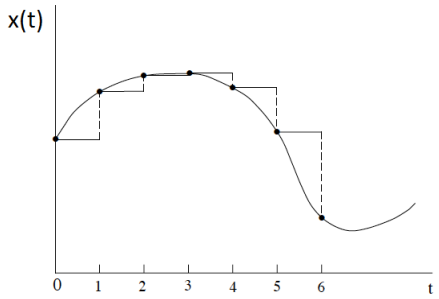


Figure 7.4: Controller symbols $\tilde{x}[1], \tilde{x}[2], \ldots$ produced by the generator by sampling of the plant output signal $x(t)$ (time-triggered plant event model) (from [AK02])

Note that, since it is assumed that there is no delay in the controller, the command signal $u(t)$ issued by the actuator is *synchronized* with the signal $x(t)$ issued by the generator. In the time-triggered model, the command $u(t)$ is therefore itself *periodic*. (In Figure 7.4, the stair length is constant and equal to τ.)

7.2.2 Quantization

Digitization has also an effect sometimes known under the name of *quantization* (see, e.g., [PB05]). Suppose that the signal u now takes its values on a *finite* domain U, instead of a dense (possibly bounded) domain or an infinite discrete domain. This means that, in Figure 7.3, the plant input signal $\mathbf{u}(t)$ is a staircase signal that can take only a *finite* number of values. In such a situation, there are many systems (even LTI systems) for which there is no control function that ensures stabilization, i.e., convergence to a unique equilibrium point (see, e.g., [BL00]). The controller can only achieve *practical stability*, that is convergence into a bounded set. The goal is then to synthesize controllers that are capable of steering the system to within sufficiently small neighborhoods of the equilibrium. The size of the final set within which the trajectories are confined is a measure of performance of the controlled dynamics. Hence, for a quantized system, the notion of *minimal invariant* set (once a proper notion of size has been defined) is useful for describing zones of practical stability.

7.2.3 Switching

A *switched system* is a digital quantized control system which consists of a finite family of continuous subsystems and a rule that orchestrates the switching between them. More precisely, we have;

Definition 31. *A* switched system *is a quadruple* $\mathscr{S} = (\mathbb{R}^n, U, \mathscr{U}, \mathscr{F})$*, where* $\mathbb{R}^n$ *is the state space;* $U = \{1, \ldots, N\}$ *is the finite set of modes;* $\mathscr{U}$ *is the set of piecewise constant functions from* $\mathbb{R}_{\geq 0}$ *to* U*, continuous from the right;* $\mathscr{F} = \{f_1, \ldots, f_N\}$ *is a collection of smooth vector fields indexed by* U*.*

A *switching signal* of $\mathscr{S}$ is a function $\mathbf{u} \in \mathscr{U}$. A piecewise $\mathscr{C}^1$ function $\mathbf{x} : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ is said to be a *trajectory* of $\mathscr{S}$ if it is continuous and there exists a switching signal $\mathbf{u} \in \mathscr{U}$ such that, at each $t \in \mathbb{R}_{\geq 0}$ where the function $\mathbf{u}$ is continuous, $\mathbf{x}$ is continuously differentiable and satisfies:

$$\dot{\mathbf{x}}(t) = f_{\mathbf{u}(t)}(\mathbf{x}(t)).$$

The times at which the switching signal changes its values are called the *switching instants*.

The scheme of switched systems is represented in Figure 7.5. It is easy to see that quantized discrete-time LTI systems is a particular subclass of switched system (for which the function $f_{\mathbf{u}(t)}(x(t))$ is of the form $Ax(t)+B\mathbf{u}(t)$). However, the class of switched systems is much more general.

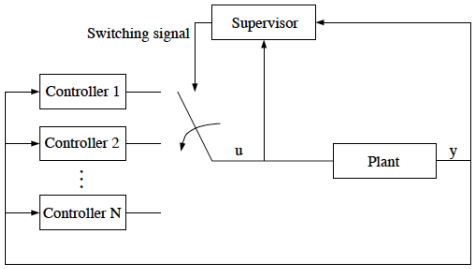


Figure 7.5: Scheme of a switching controller feedback controller

In recent years, control techniques based on switching between different controllers, as depicted in Figure 7.5, have been used in order to achieve stability and improve transient response. The importance of such control methods also stems from the existence of systems that cannot be stabilized by a single continuous feedback law (see [Bro83]). In contrast, even if the different components of a switched system working in their proper mode have no (common) equilibrium, it is still possible to control the global system in order to make its behavior similar to those of conventional stable systems near equilibrium (see, e.g., [BRC05]). Switched systems have thus found numerous applications in switching power converters and many other fields (see [LM99]).

Caveat.

Note however it is possible for a switched system to be unstable even when all the subsystems are stable around a common equilibrium point. This is true even when the subsystems are linear, as illustrated in the following example (see [AK02]). Consider the switched system $\dot{x}(t) = A_u x(t)$ where $x \in \mathbb{R}^2$, $u \in \{1,2\}$, and

$$A_1 = \begin{pmatrix} -1 & 100 \\ 10 & -1 \end{pmatrix}, A_2 = \begin{pmatrix} -1 & -10 \\ 100 & -1 \end{pmatrix},$$

with a (state-triggered) switching signal which applies A_1 (resp. A_2) when x is in the second and fourth (resp. first and third) quadrants. Both A_1 and A_2 are stable since their eigenvalues $\lambda_{1,2} = -1 \pm j\sqrt{1000}$ have negative real parts. However their trajectories are unstable (see Figure 7.6). Such a phenomenon happens because the intervals between the switchings of the dynamics decrease to 0 as time goes to infinity. This can be avoided by imposing a minimum duration (called *dwell time*) between two switching instants. This can be easily enforced for the class of *sampled* switched systems that we study in this thesis for which switchings occur with a fixed period τ.

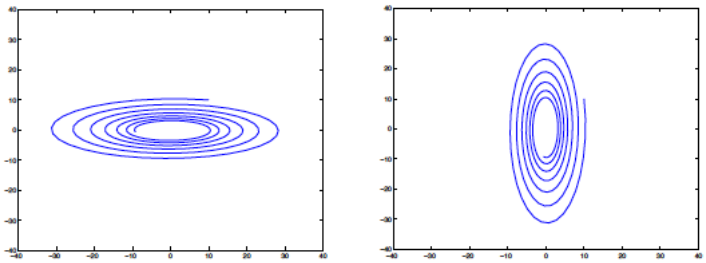


Figure 7.6: Unstable trajectory of switched system consisting of stable subsystems (from [AK02])

7.3 Control of Switched Systems Using Invariant Sets

We now consider the problem of synthesizing controllers for switched systems. This amounts to finding a switching signal that controls the system in order to satisfy some given properties. We focus on the safety and stability properties. We explain that the controller synthesis problem is related to the construction of controlled invariant sets.

7.3.1 Controlled Invariants

Given a dynamic system, a subset $\mathscr{I}$ of the state space is said to be *invariant* if it has the property that, if it contains the system state at some time, then it will contain it also in the future [Bla99].[1] We have:

$$x(t) \in \mathscr{I} \Rightarrow x(t') \in \mathscr{I}, \text{ for all } t' \geq t.$$

The concept of invariance can be easily extended to the case in which a control input is present. In this case we say that a set R is *controlled invariant* if, for all initial conditions chosen in R, we can keep the trajectory inside $\mathscr{I}$ *by means of a proper switching signal.* Let us now explain why controlled invariants are useful for proving *safety* and *stability* properties of a switched system.

7.3.2 Safety Control Problem

The safety property is typically encoded as a subset S of the continuous state space, called *safe set.* In a simple formulation, S is a box set given by the min-

[1] This property is often called "positively invariant" instead of just "invariant" in the literature.

imum and maximum values tolerated for each state variable. The associated safety properties suffice to describe typical requirements of Direct Current to Direct Current (DC-DC) power converters such as voltage regulation, current limitation, maximal current and voltage ripple.

Safety control problem: given a safe set S, determine if there exists a switching signal $\mathbf{u}$ such that if $x(0) \in S$ then $x(t) \in S$ for $t \geq 0$.

Several approaches [ABD+00, TLS00] have been proposed to solve the safety control problem. The idea of these approaches is to obtain a *controlled invariant* W which is included into S for an appropriate switching signal $\mathbf{u}$. If such a set W exists and if the initial state is in W, then the system is ensured to stay in W, hence in the safe set S. In [ABD+00], an abstract algorithm is proposed to synthesize controlled invariants using a backward iterative computation of reachable states. Furthermore the set W computed is the *maximal controlled invariant* subset of S (it contains all other controlled invariants included into S). In [TLS00], the controller synthesis problem is formulated as a game between controller and disturbance. One can then find Hamilton-Jacobi equations whose solutions describe the boundaries of the maximal safe set, and derive an associated maximally permissive controller. In Chapter 7.5, we give methods to synthesize safety controllers that are adapted to the simpler context of sampled switched systems that we consider here.

7.3.3 Stability Control Problem

Given a certain region R, many controlled invariants subsets of R exist. If, instead of looking for maximal invariant subsets, we look for finding invariants of size as small as possible around a given operating point, we get a characterization of a controller with the smallest deviation from the point, and obtain a steady-state behavior with "minimum ripple" (see [SEK03]). When periodic solutions of the system exist, we should be able to synthesize a *stability controller* that makes the trajectories converge to such periodic solutions of the system, also called *limit cycles*.

Stability control problem: given a region R, determine a switching signal $\mathbf{u}$ that makes the trajectories starting in R converge to a subregion as small as possible, ideally a limit cycle.

In Chapter 8, we give a method based on a procedure of state space decomposition, and iterated computation of forward reachable states for synthesizing stability controllers.

7.3.4 Other Controllers

We will also give some hints to solve the problem of synthesizing *robust safety* controllers that maintain the plant in a safety region in presence of disturbance or uncertainty, as well as *reachability controllers*, which drive the plant in finite time from an initial operating region to a desired one (see Chapter 8.6).

7.4 Sampled Switched Systems

In the rest of this thesis, we will focus on $\mathscr{S}^2$-systems .

7.4.1 Model

For an $\mathscr{S}^2$-system of sampling period τ, the control synthesis problem then amounts to finding the value of the switching signal at times τ, 2τ, ... In addition, we make assumptions that are commonly met in practice in embedded control applications of power electronics and automotive industry. They are as follows:

(A1) We focus on *affine* dynamics: the function $f_u(x)$ is of the form $A_u x + b_u$.

(A2) We consider that the solution of the differential equation is *continuous*: there are no "jump" of the trajectory at the switching instants.

(A3) We only consider the properties of the system at *switching instants* τ, 2τ, ..., and ignore possible state constraint violations of the system in between.

These assumptions are classically done in power electronics systems. In power electronics, a typical circuit is a network of electrical components selected from the following three groups: ideal voltage or current sources, linear elements (e.g., resistors, capacitors, inductors, transformers), and nonlinear elements acting as switches (see [SEK03]). At this level of abstraction, the behavior of a switch is idealized as having two discrete states: an open circuit and a short circuit. In a circuit with K switches, there are 2^K possible modes. In practice however, not all these modes are admissible. Some of them are not feasible because of the physical characteristics of the switches, while others are banned by the designer because of safety considerations. Because of the restricted choice of circuit elements, the resulting systems have the desirable property that the continuous dynamics of each mode are linear or affine, which justifies (A1). The absence of "jump" assumed in (A2) is met in practice because of the continuity of the laws of physics. Finally, from this continuity, it follows that the

constraint violations between switching instants are limited and become negligible for sufficiently small sampling periods.

Formally, we have (see [Tab09, Gir10]):

Definition 32. *An $\mathscr{S}^2$-system Σ of sampling period $\tau \geq 0$ is a switched system $\Sigma = (\mathbb{R}^n, U, \mathscr{U}, \mathscr{F})$ where*

- *the switching instants of $\mathbf{u} \in \mathscr{U}$ occur periodically at times $\tau, 2\tau, \ldots$*
- *$\mathscr{F}$ is a set of functions $\{f_u\}_{u\in U}$ with, for any $u \in U$ and $x \in \mathbb{R}^n$, $f_u(x) = A_u x + b_u$ with A_u a matrix of $\mathbb{R}^{n\times n}$ and b_u an array of $\mathbb{R}^n$.*

In the following, it is convenient to assume that the matrix A_u governing the dynamics of mode u, is invertible. This assumption is met in realistic models of physical systems.[2]

Given an initial condition $x_0 \in \mathbb{R}^n$ (such that $\mathbf{x}(0) = x_0$), the trajectory is fully determined by the values $u_1, u_2, \ldots$ of u at switching instants $\tau, 2\tau, \ldots$. These values define a switching signal $\mathbf{u}(t)$, which is constant on each interval $[k\tau, (k+1)\tau)$, for all $k \in \mathbb{N}$. Between two switching instants, the system is governed by a differential equation of the form: $\dot{\mathbf{x}}(t) = A_u\mathbf{x}(t) + b_u$ with $u \in U$. We will use $\mathbf{x}(t, x, u)$ to denote the point reached by Σ at time t (since last switching) under mode u from the initial condition x. This gives a transition relation $\rightarrow_u^\tau$ defined, for all x and x' in $\mathbb{R}^n$, by:

$$x \rightarrow_u^\tau x' \text{ iff } \mathbf{x}(\tau, x, u) = x'.$$

For a given $\mathscr{S}^2$-system Σ, the transition relation $\bigcup_{u\in U} \rightarrow_u^\tau$ will be denoted by $\rightarrow^\tau$. (In other words: $x \rightarrow^\tau x'$ means $x \rightarrow_u^\tau x'$ for some $u \in U$.)

Definition 33. *Given an $\mathscr{S}^2$-system Σ, a set $X \subseteq \mathbb{R}^n$ is* controlled invariant *if:*

$$\forall x \in X\ \exists u \in U\ \exists x' \in X\ x \rightarrow_u^\tau x'.$$

The set of successors of X via mode u, *denoted by $Post_{u,\tau}(X)$, or more simply by $Post_u(X)$, is:*

$$\{x' \mid x \rightarrow_u^\tau x' \text{ for some } x \in X\}.$$

The set of predecessors of X via mode u, *denoted by $Pre_{u,\tau}(X)$, or more simply by $Pre_u(X)$, is:*

$$\{x' \mid x' \rightarrow_u^\tau x \text{ for some } x \in X\}.$$

*Given a set $R \subset \mathbb{R}^n$, a subset X of R is R-*invariant via mode *u if: $Post_u(X) \subseteq R$.*

[2] For example, if the matrix associated with a model of electrical circuit is non-invertible, it is often because some resistances have been neglected and idealized to 0.

Proposition 6. *The mappings* $Post_u : 2^{\mathbb{R}^n} \to 2^{\mathbb{R}^n}$ *and* $Pre_u : 2^{\mathbb{R}^n} \to 2^{\mathbb{R}^n}$ *, with* $u \in U$*, are affine transformations.*

Proof. Given a mode $u \in U$, we have that $\dot{\mathbf{x}}(t) = A_u\mathbf{x}(t) + b_u$ with $u \in U$ and $(A_u, b_u) \in \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times 1}$. Therefore, we have that $\mathbf{x}(\tau, x, u) = e^{A_u\tau}x + \int_0^\tau e^{A_u(\tau-t)}b_u dt = C_u x + d_u$ with $C_u = e^{A_u\tau}$ and $d_u = \int_0^\tau e^{A_u(\tau-t)}b_u dt = (e^{A_u\tau} - \mathscr{I}_n)A_u^{-1}b_u$ where $\mathscr{I}_n$ denotes the identity matrix. Hence, $Post_u$ is an affine transformation. The proof is similar for Pre_u. □

As stated in assumption (A3), we will focus on the states reached by the continuous-time trajectories at switching instants $0, \tau, 2\tau, \ldots$, and disregard the continuous portions of trajectories between two switching instants. This is depicted in Figure 7.7: we focus only on the segment $F = [x_1(0), x_2(0)]$ and its exact segment successor $[x_1(\tau), x_2(\tau)]$ at time τ, which corresponds to an affine image of the form $C_u(F) + d_u$, and do not construct a polyhedral over-approximation of the continuous trajectories starting from F (see steps (b) and (c)) as in [ABD+00]. Such a restriction allows us to simplify many works of the literature. The price to be paid is that, *between* two switching instants, the system may violate temporarily a desired property.

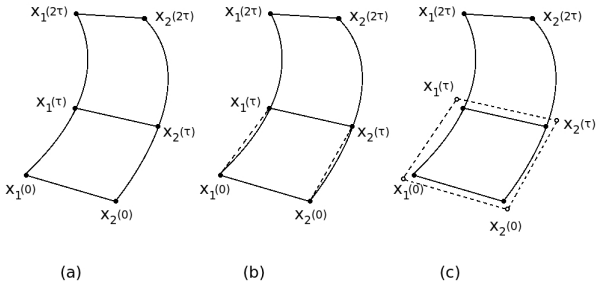


Figure 7.7: (a) A segment $F = [x_1(0), x_2(0)]$ and its exact segment successor $[x_1(\tau), x_2(\tau)]$ at time τ. (b) Approximating the set of continuous trajectories starting from F during τ time by convex hull (c) Bloating the convex polyhedron to obtain a polyhedral over-approximation (from [ABD+00])

An $\mathscr{S}^2$-system can be thus seen a *discrete-time* system governed by an affine equation of the form: $x(t+\tau) = C_u x(t) + d_u$ with $C_u \in \mathbb{R}^n \times \mathbb{R}^n$ and $d_u \in \mathbb{R}^n$. Given an $\mathscr{S}^2$-system Σ and an initial point, the set of points $\{x_0, x_1, x_2, \ldots\}$ corresponding to the states of the system at instants $0, \tau, 2\tau, \ldots$ will be referred to as the *discrete trajectory* of Σ starting at x_0. In the figures, for facilitating visualization, consecutive points of discrete trajectories will be linked together by *straight* lines (unlike the real continuous-time trajectory portion which is exponential).

Unless otherwise stated, the notation $\|\cdot\|$ will denote the Euclidean norm: for any $x \in \mathbb{R}^n$, $\|x\|$ is defined by $\|x\| = (x_1^2 + \cdots + x_n^2)^{\frac{1}{2}}$ where x_i is the ith component of the vector x. The exponential of any matrix $A \in \mathbb{R}^{n\times n}$ is denoted by e^A and is the analytic function $\sum_{i=0}^{\infty} \frac{1}{i!} A^i$. The ball of radius $\varepsilon \in \mathbb{R}_{\geq 0}$ centered at $x \in \mathbb{R}^n$ is denoted by $\mathscr{B}(x,\varepsilon)$ and is defined as the set of all the points $x' \in \mathbb{R}_{\geq 0}$ satisfying $\|x - x'\| \leq \varepsilon$. Likewise: $\mathscr{B}(X,\varepsilon) = \bigcup_{x\in X} \mathscr{B}(x,\varepsilon)$ for all $X \subseteq \mathbb{R}^n$.

Definition 34. *We say that a mode $u \in U$ is* contractive *if there exists $0 \leq \beta_u < 1$ such that, for all $x, y \in \mathbb{R}^n$:*

$$\|\mathbf{x}(\tau, x, u) - \mathbf{x}(\tau, y, u)\| \leq \beta_u \|x - y\|.$$

Given a subset $R \subset \mathbb{R}^n$, We say that a mode $u \in U$ is locally contractive in R *if there exists $0 \leq \beta_u < 1$ such that, for all $x, y \in R$: $\|\mathbf{x}(\tau, x, u) - \mathbf{x}(\tau, y, u)\| \leq \beta_u \|x - y\|$.*

It is easy to see that a mode u is contractive iff $\|e^{A_u \tau}\| = \beta_u$ for some $0 \leq \beta_u < 1$. This is equivalent to say that all the eigenvalues of $A_u\tau$ have negative real parts. Likewise, a mode u is locally contractive in R if there exists $0 \leq \beta_u < 1$ such that, for all $x \in R$, $\|e^{A_u\tau} x\| \leq \beta_u \|x\|$.[3] In order to show that a mode u is locally contractive in R, it suffices to show that there exists a right cone $\mathscr{C}$ that contains R such that $\exists 0 \leq \beta_u < 1, \forall x \in \mathscr{S}(0_{\mathbb{R}^n}, 1) \cap \mathscr{C}$, $\|e^{A_u\tau} x\| \leq \beta_u$ where $\mathscr{S}(0_{\mathbb{R}^n}, 1)$ is the unity sphere (i.e., $\mathscr{S}(0_{\mathbb{R}^n}, 1) = \{x \in \mathbb{R}^n, \|x\| = 1\}$).

A *pattern* of the form $(u_1 \cdot u_2 \cdots u_m)$ is a finite sequence of modes $u_1, u_2, \ldots, u_m$ of U. A *k-pattern* is a pattern of length at most k. Patterns will be often associated with finite paths in oriented graphs whose edges are labeled by modes. We will use the expression π^i for denoting the concatenation of i patterns equal to π, and π^* for the concatenation of π an arbitrary number of times.

The definition of successors via modes (Definition 33) extends naturally for patterns. Formally, for $X \subseteq \mathbb{R}^n$ and a pattern π of the form $(u_1 \cdot u_2 \cdots u_m)$, we have: $Post_\pi(X) = Post_{u_m}(\cdots(Post_{u_1}(X))\cdots)$. We write sometimes $x \rightarrow_\pi x'$ to mean $x \rightarrow^\tau_{u_1} x_1 \rightarrow^\tau_{u_2} \cdots x_{m-1} \rightarrow^\tau_{u_m} x'$ for some $x_1, \ldots, x_{m-1} \in \mathbb{R}^n$.

Likewise, definitions of predecessors, R-invariance, (local) contractivity for modes extend naturally to those for patterns. From Proposition 6, it follows:

Proposition 7. *Let π be a pattern. Then the mappings $Post_\pi : 2^{\mathbb{R}^n} \rightarrow 2^{\mathbb{R}^n}$ and $Pre_\pi : 2^{\mathbb{R}^n} \rightarrow 2^{\mathbb{R}^n}$ are affine transformations.*

The image of a convex set X by $Post_\pi$ (resp. Pre_π) is therefore a convex set. Given a convex set $R \subset \mathbb{R}^n$ and a pattern π, in order to show that a convex subset X is R-invariant via π, it suffices to show that every vertex of X is mapped via $Post_\pi$ to a point of R.

[3] Note that this requires that at least one eigenvalue of $A_u\tau$ has a negative real part.

7.4.2 Illustrative Examples

Example 12. *(**Boost DC-DC Converter**). This example is taken from [BPM05] (see also, e.g., [GPT10, BRC05, SEK03]). This is a boost DC-DC converter with one switching cell (see left part of Figure 7.8). The state of the system is* $x(t) = [i_l(t)\ v_c(t)]^T$ *where* $i_l(t)$ *is the inductor current, and* $v_c(t)$ *the capacitor voltage. There are two operation modes depending on the position of the switching cell. When the switch is close (mode 2), the inductor current* i_l *increases and energy is stored into the inductance. When the switch is open (mode 1), the energy accumulated in the inductance is transferred into the capacitor. The dynamics*

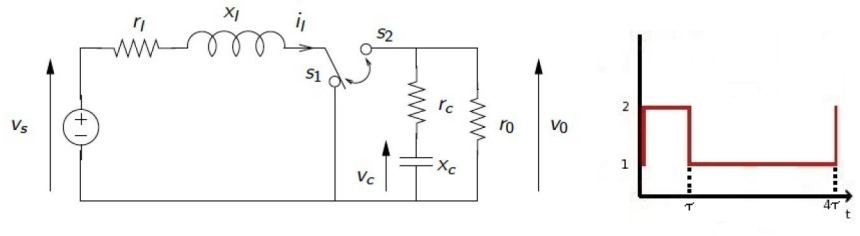


Figure 7.8: Left: scheme of the boost DC-DC converter; right: cell switching for pattern $(2 \cdot 1 \cdot 1 \cdot 1)$

associated with mode u *is of the form* $\dot{x}(t) = A_u x(t) + b_u$ $(u = 1,2)$ *with*

$$A_1 = \begin{pmatrix} -\frac{r_l}{x_l} & 0 \\ 0 & -\frac{1}{x_c}\frac{1}{r_0+r_c} \end{pmatrix} \quad b_1 = \begin{pmatrix} \frac{v_s}{x_l} \\ 0 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} -\frac{1}{x_l}(r_l + \frac{r_0 r_c}{r_0+r_c}) & -\frac{1}{x_l}\frac{r_0}{r_0+r_c} \\ \frac{1}{x_c}\frac{r_0}{r_0+r_c} & -\frac{1}{x_c}\frac{1}{r_0+r_c} \end{pmatrix} \quad b_2 = \begin{pmatrix} \frac{v_s}{x_l} \\ 0 \end{pmatrix}$$

We will use the numerical values of [BPM05], expressed in the per unit system: $x_c = 70$, $x_l = 3$, $r_c = 0.005$, $r_l = 0.05$, $r_0 = 1$, $v_s = 1$. *The sampling period is* $\tau = 0.5$.

A general goal of the control is to stabilize the output voltage v_0 *around a desired value* v_e. *The range of variations of the output voltage and inductor current should be limited in order to avoid phenomena of inductor saturation and blocking voltage stress of the switch. This corresponds to the specification of a safety area* S. *The safety control problem is to find a strategy for deciding which sequence of patterns to apply in order to keep the state within* S. *An example of pattern of length* 4 *is illustrated on the right part of Figure 7.8: it corresponds to the application of mode 2 on* $(0,\tau]$ *and mode 1 on* $(\tau, 4\tau]$. *The control can be state-independent, consisting in the repeated application of the same sequence of patterns (computed off line), or state-dependent, with the application of a pattern depending on the current value of the electrical state.*

Example 13. *(**Two-Room Building Heater**). This example is taken from [Gir12]. It is a simple thermal model of a two-room building. One of the room can be heated via a heating device. The two rooms communicate such that heat from one room can diffuse to the other. Moreover, the rooms are surrounded by an environment that has a fixed temperature. By controlling when to turn on and off the heating device, one is interested in maintaining the two rooms at a comfortable temperature. Let $T = [T_1\ T_2]^T$ be the state variable, where T_i is the temperature of room i $(i = 1,2)$. The dynamics of the system are given by the following equation:*

$$\dot{T} = \begin{pmatrix} -\alpha_{21} - \alpha_{e1} - \alpha_f u & \alpha_{21} \\ \alpha_{12} & -\alpha_{12} - \alpha_{e2} \end{pmatrix} T + \begin{pmatrix} \alpha_{e1} T_e + \alpha_f T_f u \\ \alpha_{e2} T_e \end{pmatrix}$$

where u is a mode of value 0 or 1, and the heat transfer coefficients and external temperatures are given by the values: $\alpha_{12} = 5.10^{-2}$, $\alpha_{21} = 5.10^{-2}$, $\alpha_{e1} = 5.10^{-3}$, $\alpha_{e2} = 3.3.10^{-3}$, $\alpha_f = 8.3.10^{-3}$, $T_e = 10$, $T_f = 50$. The sampling period is $\tau = 5$.

Example 14. *(**Helicopter Motion**). This example is taken from [DLHT11]. The problem is to control a quadrotor helicopter to some position on top of a stationary ground vehicle, while satisfying constraint on the relative velocity. By controlling the pitch and roll angles, one can modify the speed and the position of the helicopter. A typical problem is to find a switching rule, depending on the position and velocity of the helicopter, in order to keep the system state within a safe area, avoiding excessive speed or distance to the ground vehicle. Let g be the gravitational constant, x (resp. y) the position according to x-axis (resp. y-axis), $\dot{x}$ (resp. $\dot{y}$) the velocity according to x-axis (resp. y-axis), ϕ the pitch command, ψ the roll command. The possible commands for the pitch and the roll are the following: $\phi, \psi \in \{-10, 0, 10\}$. Since each mode corresponds to a pair (ϕ, ψ), there are 9 modes. The dynamics of the system are given by the equation:*

$$\dot{X} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} X + \begin{pmatrix} 0 \\ g\ \sin(-\phi) \\ 0 \\ g\ \sin(\psi) \end{pmatrix}$$

where X is $[x\ \dot{x}\ y\ \dot{y}]^T$. The sampling period is $\tau = 0.1$. Since the variables x and y are decoupled in the equations and follow the same equations (up to the sign of the command), it suffices to study the control for x. (The control for y is opposite.)

7.4.3 Zonotopes

The construction of invariant sets under the form of (union of) *polyhedral* sets is natural because polyhedra correspond to sets of linear constraints of the state

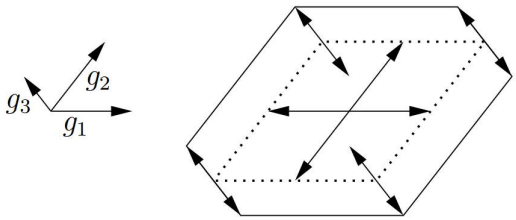


Figure 7.9: Example of a zonotope with three generators (taken from [Gir05])

space. Furthermore, they are well-suited to the approximation of reachability sets and domains of attractions of dynamic systems. *Zonotopes* are a data structure which is very useful for representing and manipulating efficiently convex polytopes (see, e.g., [Küh98, Gir05, ASB08]). They can be seen as symmetric polyhedra where a facet must be parallel to an opposing facet (see Figure 7.9 for the illustration of a zonotope with 3 generators).

The class of zonotopes is closed under linear transformation and Minkowski sum[4]. Furthermore, using zonotopes, it is easy to introduce uncertainty or disturbance in the dynamics of the models. A zonotope is defined by a center c to which linear segments $l_i = \beta^{(i)} \cdot g^{(i)}$, $-1 \leq \beta^{(i)} \leq 1$ are added via Minkowski sum.

Definition 35. *A* zonotope *is a set*

$$Z = \{x \in \mathbb{R}^n : x = c + \Sigma_{i=1}^{p} \beta^{(i)} \cdot g^{(i)},\ -1 \leq \beta^{(i)} \leq 1\}$$

with $c, g^{(1)}, \ldots, g^{(p)} \in \mathbb{R}^n$.

The vectors $g^{(1)}, \ldots, g^{(p)}$ are referred to as the *generators* and c as the *center* of the zonotope. It is convenient to represent the set of generators as an $n \times p$ matrix G of columns $g^{(1)}, \ldots, g^{(p)}$. The notation is $< c, G >$.

A *box* (or *rectangle*) is a cartesian product of n closed intervals. It can be seen as a zonotope of the form $< c, D >$ where c is the center of the box, and D is an $n \times n$ diagonal matrix whose (i,i)th element is equal to half the size of the ith interval, for $1 \leq i \leq n$. Boxes play an important role in invariance theory (see, e.g., [PB08, ATS09]).

The smallest box containing a zonotope $Z =< c, G >$ is called the *bounding box of* Z, and denoted by $\square(Z)$. We have: $\square(Z) =< c, D >$ where D is an $n \times n$ diagonal matrix whose (i,i)th element is equal to $\Sigma_{\ell=1}^{p} |G_{i,\ell}|$, for $1 \leq i \leq n$.

Given a zonotope $Z =< c, G >$, the transformation of Z via an affine function $x \mapsto Cx + d$ is a zonotope of the form $< Cc + d, CG >$. The successor set $Post_u(Z)$

[4]The Minkowski of two sets A, B is defined by $A + B = \{a + b \mid a \in A, b \in B\}$.

of Z via a mode u can thus be simply computed using zonotopes, using matrix multiplication whose complexity is (at most) cubic.

Zonotopes allow to compute easily overapproximations of the successor sets in order to take into account small perturbations (or uncertainties) of the system dynamics. All the dynamics of the system are now of the form $\dot{x}(t) = A_u x(t) + b_u + \varepsilon(t)$ where $\varepsilon(t)$ represents disturbance under the form of a vector belonging to a given box $\Lambda = [-\varepsilon_1, +\varepsilon_1] \times \cdots \times [-\varepsilon_n, +\varepsilon_n]$ of $\mathbb{R}^n$, with $\varepsilon_i \geq 0$ for $i = 1, \ldots, n$. We will use $\mathbf{x}(t, x, u, \varepsilon)$ to denote the point reached by the system at time t under mode u with disturbance ε, from the initial condition x. This entails a transition relation $\rightarrow_\tau^{u,\varepsilon}$ defined, for all $x, x' \in \mathbb{R}^n$, $\varepsilon \in \mathbb{R}^n_{\geq 0}$ and $u \in U$ by:

$$x \rightarrow_\tau^{u,\varepsilon} x' \text{ iff } \mathbf{x}(\tau, x, u, \varepsilon) = x'.$$

Given a box $\Lambda = [-\varepsilon_1, +\varepsilon_1] \times \cdots \times [-\varepsilon_n, +\varepsilon_n]$ of $\mathbb{R}^n$, we define $Post_u(X, \Lambda) = \{x' \mid \exists \varepsilon \in \Lambda,\ x \rightarrow_\tau^{u,\varepsilon} x'\}$. This definition of $Post_u$ with perturbation naturally extends to $Post_\pi$ where π is a pattern. If X is given under the form of a zonotope $< c, G >$, then it is easy to compute an overapproximation of $Post_u(X, \Lambda)$. Suppose that the successor set *without* perturbation $Post_u(X)$, is an affine transformation of the form $CX + d$. We have:

Lemma 12. *Consider a zonotope $X =< c, G >$, a box $\Lambda = [-\varepsilon_1, +\varepsilon_1] \times \cdots [-\varepsilon_n, +\varepsilon_n]$ of $\mathbb{R}^n$. We have:*

$$Post_u(X, \Lambda) \subseteq < Cc + d, \begin{pmatrix} CG & \tau D_\lambda \end{pmatrix} >,$$

with: $D_\Lambda = \begin{pmatrix} \varepsilon_1 & 0 & \ldots & 0 \\ 0 & \varepsilon_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \varepsilon_n \end{pmatrix}$.

The process can be iterated to compute a zonotopic overapproximation of $Post_\pi(X, \Lambda)$, for any pattern π. This technique will be used in Chapter 8.6 in order to handle the problem of *robust safety* control and the problem of *nonlinear dynamics*.

7.5 Safety Controllers

In this section, we are interested in finding controllers of an $\mathscr{S}^2$-system which make the variables of the system stay within the limits of a given area S. The problem amounts to finding a switching rule that selects a mode ensuring that the system will still be in S at the next sampling time, and so on iteratively. If

we consider S as a predefined *safe* region for the operating states of the system, then such a switching rule can be seen as a *safety controller*. The problem is closely related to the problem of finding a controlled invariant subset of S. It is interesting to design a controller that is as permissive as possible since this allows to formulate secondary control objectives in order to satisfy various performance criteria inside the safe set. This amounts to synthesizing a controlled invariant subset of S which is as large as possible, ideally *maximal*.

We present a *direct* approach working on the original continuous state space, which makes use of a general procedure of backward fixed point computation and an *indirect* approach working on an abstract discrete state space, which makes use of the notion of approximate bisimularity. The two methods are applied to the 3-cell converter application of power electronics.

7.5.1 Backward Fixed Point Computation (Direct Approach)

Suppose we are given an $\mathscr{S}^2$-system Σ and a set S of *safe* states. Let us consider a very general approach for synthesizing a (maximal) subset S^* of S which is *controlled invariant* (i.e., such that: $\forall x \in S^* \ \exists u \in U = \{1 \dots N\} \ \exists x' \in S^* \ x \rightarrow_u^\tau x'$). Given a set S, it is easy to show that the union of two controlled invariant subsets of S is itself a controlled invariant subset of S. Accordingly, the notion of *maximal controlled invariant* subset of S is well-defined and corresponds to the union of all the controlled invariant subsets of S.

Consider the operator $F_S : 2^{\mathbb{R}^n} \rightarrow 2^{\mathbb{R}^n}$ defined by:

$$F_S(X) = \bigcup_{u=1\dots N} Pre_u(X) \cap S.$$

The set $F_S(X)$ contains all the states $x \in X \cap S$ for which the successors via u of x are in X. The next result states that a *maximal fixed point* of F_S exists.

Proposition 8. *The operator $F_S : 2^{\mathbb{R}^n} \rightarrow 2^{\mathbb{R}^n}$ satisfies:*

1. *The sequence $\{F_S^i(\mathbb{R}^n)\}_{i \geq 0}$ is nested and decreasing.*
2. *The maximal fixed point S^* of F_S satisfies:*

$$S^* = \lim_{i \rightarrow \infty} F_S^i(\mathbb{R}^n) = \bigcap_{i \geq 0} F_S^i(\mathbb{R}^n);$$

3. *The maximal fixed point S^* of F_S is the maximal controlled invariant subset of S.*

Proof. See [Tab09, Chapter 6] for a proof. □

The maximal fixed point S^* of F_S (i.e., the maximal solution of equation $X = \bigcup_{u=1..N} Pre_u(X) \cap S$) can be computed by iteration of F_S until a fixed point S^* is obtained. The general algorithm is of the form:

Algorithm 6: Synthesis of maximal controlled invariant subset

Input: A set $S \subset \mathbb{R}^n$
Output: A maximal invariant subset S^* of S

$X^0 := S$
repeat
$X^{k+1} := X^k \cap \bigcup_{u=1..N} Pre_u(X^k)$
until $X^{k+1} = X^k$
$S^* = X^k$

The correctness of Algorithm 6 relies on the fact that, at step $k \geq 0$, the set X^k is the set of starting points of trajectories of length k contained in S. Formally: $X^k = \{x \in S \mid x \rightarrow_\tau x_1 \rightarrow_\tau \cdots \rightarrow_\tau x_k \text{ for some } x_1, \ldots, x_k \in S\}$. It follows that S^* is the set of starting points of infinite trajectories contained in S, which means that S^* is the maximal invariant subset of S.

Henceforth, we suppose that the input S is given under a polyhedral form. Every set X^k can be put under the form of a finite union of polyhedral components: each polyhedral component P of X^k is obtained as $Pre_u(Q) \cap R$, where u is in U, and Q and R are themselves two polyhedral components of X^{k-1}. (Recall that the operator Pre_u is an affine transformation which maps a polyhedron into another one; see Chapter 7.4). The test $X^{k+1} = X^k$ is performed by testing if the vertices of the components of X^k belong to (components of) X^{k+1}, and vice versa. If the test succeeds, Algorithm 6 terminates, and outputs a set S^* which is a union of polyhedral components.

Let us now explain how one can derive a state-dependent control strategy that allows to maintain the system always in S^*, using a simple additional information storage in algorithm 6. We modify the algorithm as follows: for each polyhedron component P produced at step k (of the form $Pre_u(Q) \cap R$, with $Q, R \subset X^{k-1}$), we store the mode u with which it has been produced. If Algorithm 6 terminates, the output S^* is given under the form of a finite set of polyhedral components together with their associated modes. The control is now as follows: when at a switching time, the system state lies in a component P of S^*, one applies the associated mode u of P; at the next switching time, the system lies in a component P' of S^*, and one applies the associated mode u', and so on iteratively. Such a control which allows to stay in the maximal invariant S^* subset of S is said to be *maximally safe* (or *maximally permissive*). In a further step, one can refine the maximally permissive controller in order to satisfy various performance criteria inside the safe set.

Example 15. *To illustrate this approach, we synthesize a control for the boost DC-DC converter with one cell (see Example 12 for a description of the system). We have* $S = [3.0, 3.4] \times [1.5, 1.8]$ *in the* (i_l, v_c) *plane, and* $\tau = 0.5$. *Algorithm 6 terminates in 2 steps:*

- *at step 1, it produces two polyhedral components* $P_1 = Pre_1(S) \cap S$ *and* $P_2 = Pre_2(S) \cap S$, *with* $X^1 = P_1 \cup P_2$.
- *at step 2, the set* X^2 *produced is the union of 8 polyhedral components* $Pre_i(P_j) \cap P_k$ *with* $i, j, k \in \{1, 2\}$, *and it can be seen that* $X^2 = X^1$.

This means that $S^* = X^1 = P_1 \cup P_2$. *In Figure 7.10, the uncontrollable part* $S \setminus S^*$ *corresponds to the "horizontal" polyhedra colored in black in the lower left and upper right parts of* S. *The controlled subset* P_1 *corresponds to the "vertical" left polyhedron, and* P_1 *to the right one. If the system state is in* P_1 *(resp.* P_2*) at a switching time, then mode* 1 *(resp.* 2*) should be applied. Mode 1 or 2 can be arbitrarily applied when the system lies in* $P_1 \cap P_2$. *A controlled trajectory starting at point* $x_0 = (3.01, 1.79) \in S^*$ *is depicted in Figure 7.10 and 7.11. One can see that the trajectory stays within* $S^* \subset S$.

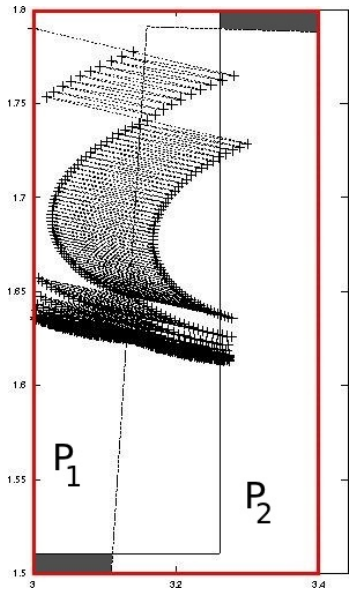


Figure 7.10: Maximal controlled invariant subset of $S = [3.0, 3.4] \times [1.5, 1.8]$, composed of two polyhedra P_1 (mode 1) and P_2 (mode 2), with a controlled trajectory starting at $x_0 = (3.01, 1.79)$

Algorithm 6 involves the computation of the predecessor operator, intersection and test of point containment for polyhedra. However, the number of

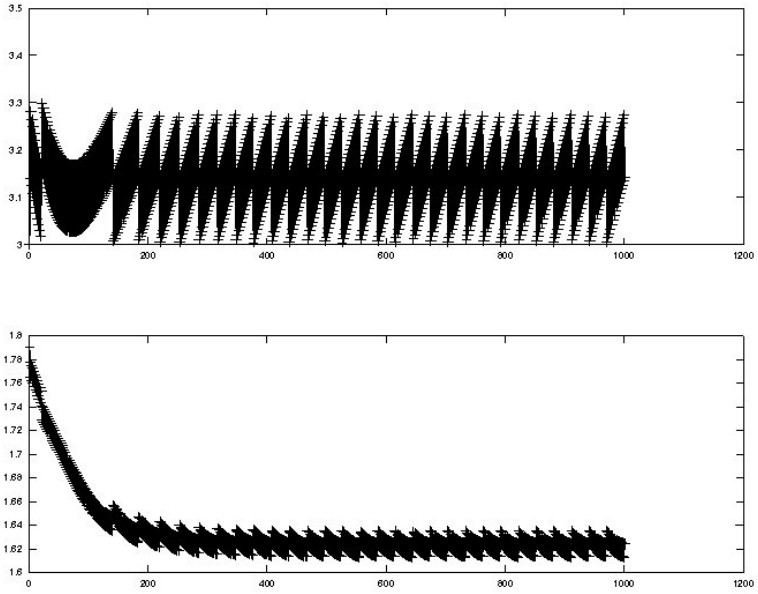


Figure 7.11: Discrete-time trajectory starting from point $x_0 = (3.01, 1.79)$, using the control found by the direct method. Above: evolution of v_c in time; below: evolution of i_l.

polyhedral components increases exponentially at each step. A realistic implementation of Algorithm 6 requires to merge different polyhedral components into an *under-approximated* polyhedral form. This can be done, using, e.g., the notion of *griddy polyhedra* (see [ABD+00]) i.e., sets that can be written as unions of closed unit hypercubes with integer vertices. The under-approximation process also helps the algorithm to terminate. The output S^* of the algorithm is then an invariant subset of S, but is no longer maximal.

7.5.2 Approximate Bisimulation (Indirect Approach)

The direct application of Algorithm 6 at the continuous state level works well on simple examples, as explained in Section 7.5.1. However, there is no guarantee of termination of the procedure because the state space is infinite. An interesting alternative approach is the "indirect approach": it consists in making an *abstraction* of the system into a finite discrete system. Algorithm 6 then always terminates, and allows to synthesize a maximally safe abstract controller, from which a controller can be derived at the real level, at the price of a certain ap-

proximation. Moreover the the switching rule can be computed *off line* while the switching rule has to be computed *on line* in the direct approach. The *indirect* approach of controller synthesis, based on finite-state approximate models, originates from [RO98]. The notion of approximate bisimulation originates from [GP05]. See also [Tab05] and [Tab09] for an exposition of several classes of hybrid systems admitting abstract models, along with the relationships between them.

We now explain such a method using the notion of *approximate bisimulation.*

In [GPT10], the authors propose a method for abstracting a switched system under the form of a discrete model, that is equivalent to the original one, under certain Lyapunov-based stability conditions. They use an Euclidean norm $\|.\|$, and define the approximation of the set of states $\mathbb{R}^n$ as follows:

$$[\mathbb{R}^n]_\eta = \{x \in \mathbb{R}^n \mid x_i = k_i \frac{2\eta}{\sqrt{n}},\ k_i \in \mathbb{Z}, i = 1, \ldots, n\},$$

where $\eta \in \mathbb{R}^+$ is a state space discretization parameter. It is then easy to see that: $\forall x \in \mathbb{R}^n\ \exists q \in [\mathbb{R}^n]_\eta : \|x - q\| < \eta$. The transition relation $\rightarrow_\tau^u$ of Σ is then approximated as follows: Let $q \in [\mathbb{R}^n]_\eta$ and $q_e = \mathbf{x}(\tau, q, u)$ such that $q \rightarrow_\tau^u q_e$ in the real system, let $q' \in [\mathbb{R}^n]_\eta$ with $\|q_e - q'\| < \eta$. Then we have $q \rightarrow_{\tau,\eta}^u q'$ for the approximated transition relation. Formally, the transition relation of the abstract system Σ_η is defined as follows.

Definition 36. *Given a switched system $\Sigma : (\tau, U, \mathscr{F})$ and its trajectory $\mathbf{x} : \mathbb{R}^+ \rightarrow \mathbb{R}^n$, the system Σ_η is the transition system $(Q, \rightarrow_{\tau,\eta}^u)$ defined by:*

- *the set of states is $Q = [\mathbb{R}^n]_\eta$*
- *the transition relation is given by*

$$q \rightarrow_{\tau,\eta}^u q' \text{ iff } \|\mathbf{x}(\tau, q, u) - q'\| \leq \eta$$

This relation is depicted on Figure 7.12 (see [Gir10]). The notion of "approximate bisimilarity" between systems Σ and Σ_η is defined as follows.

Definition 37. *Systems Σ and Σ_η are ε-*bisimilar *(or bisimilar with precision ε) if:*

1. *For all $x \in \mathbb{R}^n$ and $q, q' \in [\mathbb{R}^n]_\eta$: $(\|x - q\| \leq \varepsilon \wedge q \rightarrow_{\tau,\eta}^u q') \Rightarrow \|x' - q'\| \leq \varepsilon$ for some $x' = \mathbf{x}(\tau, x, u)$ (i.e. for some $x' : x \rightarrow_\tau^u x'$), and*
2. *For all $x, x' \in \mathbb{R}^n$ and $q \in [\mathbb{R}^n]_\eta$: $(\|x - q\| \leq \varepsilon \wedge x \rightarrow_\tau^u x') \Rightarrow \|x' - q'\| \leq \varepsilon$ for some $q' \in [\mathbb{R}^n]_\eta$ with $\|\mathbf{x}(\tau, q, u) - q'\| \leq \eta$ (i.e. for some $q' : q \rightarrow_{\tau,\eta}^u q'$).*

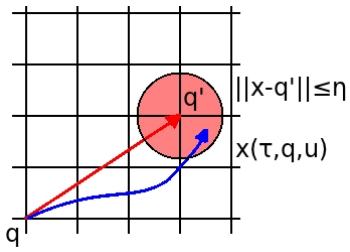


Figure 7.12: Abstract transition relation (from [Gir10])

Consider a switched system $\Sigma = (\tau, U, F)$, a desired precision ε and a time sampling value τ. Under certain Lyapunov-based stabilization conditions, it is shown in [GPT10] that there exists a space sampling value η such that the transition systems of Σ and Σ_η are approximately bisimilar with precision ε. In the context of $\mathscr{S}^2$-systems, the conditions of Lyapunov-based stabilization can be simplified as follows.

Theorem 10. *Consider a switched system $\Sigma = (\tau, U, F)$, a desired precision ε and a time sampling value τ. If all the modes of U are contractive, there exists a space sampling value η such that the transition systems of Σ and Σ_η are approximately bisimilar with precision ε.*

Proof. Because of the contractivity of the modes of Σ, we have, for all $u \in U$: $\|\mathbf{x}(\tau, x, u) - \mathbf{x}(\tau, y, u)\| \leq \beta \|x - y\|$ for some $0 \leq \beta < 1$. The proof of ε-bisimularity is based on the fact that we can choose η so that $\beta\varepsilon + \eta \leq \varepsilon$ is true (which is possible because $\beta < 1$). We have indeed:

1. $(\|x - q\| \leq \varepsilon \wedge q \rightarrow^u_{\tau,\eta} q') \Rightarrow$
   $\|x' - q'\| = \|\mathbf{x}(\tau, x, u) - q'\| \leq \|\mathbf{x}(\tau, x, u) - \mathbf{x}(\tau, q, u)\| + \|\mathbf{x}(\tau, q, u) - q'\| \leq \beta\varepsilon + \eta \leq \varepsilon$, with $x' = \mathbf{x}(\tau, x, u)$ (i.e., $x' : x \rightarrow^u_\tau x'$ for some $u \in U$)

2. $(\|x - q\| \leq \varepsilon \wedge x \rightarrow^u_\tau x') \Rightarrow$
   $\|x' - q'\| = \|\mathbf{x}(\tau, x, u) - q'\| \leq \|\mathbf{x}(\tau, x, u) - \mathbf{x}(\tau, q, u)\| + \|\mathbf{x}(\tau, q, u) - q'\|, \leq \beta\|x - q\| + \eta \leq \beta\varepsilon + \eta \leq \varepsilon$ with q' such that $q \rightarrow^u_{\tau,\eta} q'$.

□

Note that Theorem 10 holds for any norm $\|\cdot\|$. For implementing the method, it may be convenient to use the infinity norm (defined by $\|x\| = \max_{i=1\cdots n} |x_i|$, for all point $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$) rather than the Euclidean norm in order to reduce the overlapping of two adjacents bowls of radius η, and the nondeterminism of relation $\rightarrow^u_{\tau,\eta}$. Accordingly, the definition of $[\mathbb{R}^n]_\eta$ should be

$$[\mathbb{R}^n]_\eta = \{x \in \mathbb{R}^n \mid x_i = 2k_i\eta \text{ for some } k_i \in \mathbb{Z} \text{ and } i = 1, 2, \ldots, n\}.$$

Let us now explain how to apply Theorem 10 in order to synthesize a safety controller. Consider a bounded subset S of $\mathbb{R}^n$. The set $S_\eta = [\mathbb{R}^n]_\eta \cap S$ is *finite*. So, if Algorithm 6 runs with $X^0 = S_\eta$ as an input, it terminates and outputs a maximal controlled invariant subset, say S^*_η, of S_η. For each point $q \in S^*_\eta$, there exists a point $q' \in S^*_\eta$ such that $q \rightarrow^u_{\tau,\eta} q'$ sor some $u \in U$. Using the relation $\bigcup_{u\in U} \rightarrow^u_{\tau,\eta}$ restricted to S^*_η, one can define a finite state automaton $\mathscr{A}_\eta$ on S^*_η. This automaton $\mathscr{A}_\eta$ can be seen as a maximally safe controller of S_η. From such a controller, it is then possible, using the bisimilarity stated in Theorem 10, to derive a controller for the real model Σ, which keeps the switched system Σ in $\mathscr{B}(S^*, \varepsilon)$ (see [Tab08]).

An alternative approach consists in observing that, for all point q of S^*_η, there exists a quasi-cyclic sequence of transitions of $\mathscr{A}_\eta$ starting at q of the form $\pi \cdot \sigma^*$, where π and σ are finite sequences of modes. (This is because an infinite path in a finite graph should go through the same vertex twice.) For all $q \in S^*_\eta$, one can compute statically such a quasi-cyclic sequence starting at q. Let us denote it by $\varphi(q)$. This induces a safety controller for the real system Σ as follows: given a state $x \in S^*$, find a state $q \in S^*_\eta$ such that $\|x - q\| \leq \eta$; then apply the sequence $\varphi(q)$ to x. It follows from Theorem 10 that, under such a control, the state of Σ always stays in $\mathscr{B}(S^*, \varepsilon)$.

Note that the correctness of the synthesis of a safety controller for S at the continuous state level relies on Theorem 10. Actually, one can relax the assumption of contractivity of the modes of Σ made in this theorem, and just assume the *local contractivity* of modes in S. We illustrate the method on the boost DC-DC converter.

Example 16. *The method is applied on the boost converter of with the same safe set as in Example 15: $S = [3, 3.4] \times [1.5, 1.8]$ in plane (i_l, v_c), For the desired precision, we take $\varepsilon = 3.0$. It can be seen that the system is locally contractive in S with a contraction factor $\beta = 0.99202$. For the discretization parameter, we take $\eta = 1/40$ (which satisfies $\eta < \varepsilon(1-\beta)$). See Figure 7.13 for one of the connected components of the graph of the automaton $\mathscr{A}_\eta$. Each cycle in the subgraph corresponds to a periodic switching rule of the converter which ensures that the electric variables lie inside the predefined S up to ε. For example, we consider the cycle passing through vertices numbered:* $159, 243, 173, 257, 187, 271, 201, 285, 215, 299, 229, 159$. *This corresponds to the application of the cyclic sequence of modes:* $(1 \cdot 2 \cdot 1 \cdot 2 \cdot 1 \cdot 2 \cdot 1 \cdot 2 \cdot 1 \cdot 2 \cdot 2)^*$. *A controlled trajectory starting at point $x_0 = (3.0, 1.79)$, is given in Figure 7.14. The box S is delimited by the dashed line. One can see that the system largely exceeds the limits of S (but stays inside the ε-approximation $\mathscr{B}(S, \varepsilon)$).*

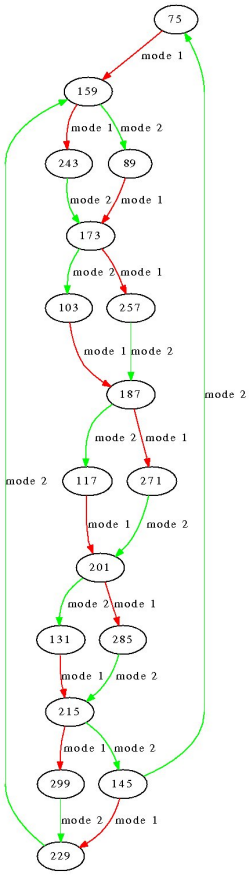


Figure 7.13: Graph of an abstract safety controller of the boost DC-DC converter, with $\eta = \frac{1}{40}$ and $S = [3, 3.4] \times [1.5, 1.8]$ obtained by a script of ours

7.5.3 Application to a 3-cells Boost DC-DC Converter

We now apply the direct and indirect methods for synthesizing safety controllers for a bigger example: a boost DC-DC converter with 3 cells. This is a real-life prototype built by the SATIE Electronics Laboratory (ENS Cachan) for the automotive industry. See Figure 7.15 for a picture of the system.

7.5.4 Model

The boost DC-DC converter with 3 cells relies on the same principle as the one with one cell. An advantage of this system is its robustness: even if one switching cell is damaged, the system is still controllable with the restricted set of modes that remain available. This system is naturally more complex: there are 4 continuous variables of interest (instead of two), and $2^3 = 8$ modes (instead of

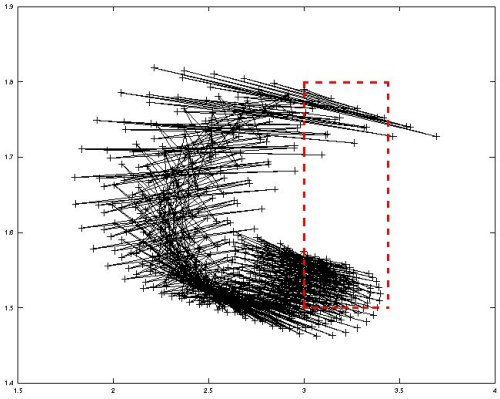


Figure 7.14: Trajectory in plane (i_l, v_c) starting at $x_0 = (3.0, 1.79)$ controlled by switching rule $(1{\cdot}2{\cdot}1{\cdot}2{\cdot}1{\cdot}2{\cdot}1{\cdot}2{\cdot}1{\cdot}2{\cdot}2)^*$ found by the indirect method (precision $\varepsilon = 3$); the dashed box corresponds to $S = [3, 3.4] \times [1.5, 1.8]$.

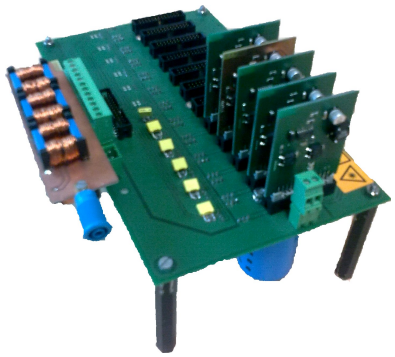


Figure 7.15: 3-cells converter built by SATIE Electronics Laboratory

two). Each mode is a triple $(\sigma_1\sigma_2\sigma_3)$ where σ_i indicates whether cell i is open ($\sigma_i = 0$) or closed ($\sigma_i = 1$). The electrical scheme is presented in Figure 7.16. An example of pattern is presented in Figure 7.17. The pattern is of the form $((100)\cdot(000)\cdot(010)\cdot(000)\cdot(001)\cdot(000))$ (or $(2\cdot1\cdot3\cdot1\cdot5\cdot1)$ under a decimal-like form), and corresponds to: $(1\cdot0\cdot0\cdot0\cdot0\cdot0)$ for σ_1, $(0\cdot0\cdot1\cdot0\cdot0\cdot0)$ for σ_2 and $(0\cdot0\cdot0\cdot0\cdot1\cdot0)$ for σ_3.

The system satisfies the following equations:

$$U\begin{pmatrix}\sigma_1\\ \sigma_2\\ \sigma_3\\ 0\end{pmatrix}+\begin{pmatrix}-2r & 0 & 0 & -1\\ 0 & -2r & 0 & -1\\ 0 & 0 & -2r & -1\\ 1 & 1 & 1 & -1/R\end{pmatrix}\begin{pmatrix}x_1\\ x_2\\ x_3\\ x_4\end{pmatrix}=\begin{pmatrix}2L & -M & -M & 0\\ -M & 2L & -M & 0\\ -M & -M & 2L & 0\\ 0 & 0 & 0 & C\end{pmatrix}\begin{pmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \dot{x}_4\end{pmatrix}$$

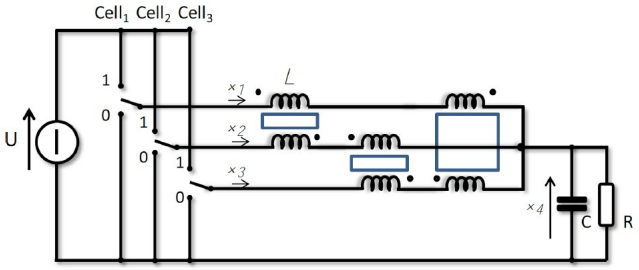


Figure 7.16: Electrical scheme of the DC-DC converter with 3 cells

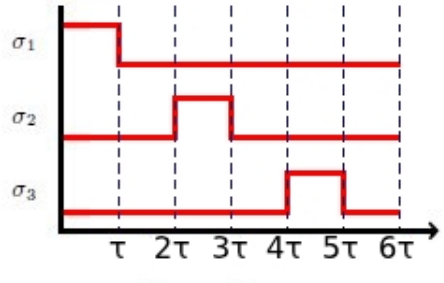


Figure 7.17: Switching rule for the 3-cells boost DC-DC converter on one period of length 6τ, $\sigma_1 = (1 \cdot 0^5)$, $\sigma_2 = (0^2 \cdot 1 \cdot 0^3)$, $\sigma_3 = (0^4 \cdot 1 \cdot 0)$, and the corresponding control pattern is $(2 \cdot 1 \cdot 3 \cdot 1 \cdot 5 \cdot 1))$

That can be rewritten to fit our framework as:

$$\dot{x} = M_{LC}^{-1} M_S x + b_\sigma$$

with

$$M_{LC} = \begin{pmatrix} 2L & -M & -M & 0 \\ -M & 2L & -M & 0 \\ -M & -M & 2L & 0 \\ 0 & 0 & 0 & C \end{pmatrix}, M_S = \begin{pmatrix} -2r & 0 & 0 & -1 \\ 0 & -2r & 0 & -1 \\ 0 & 0 & -2r & -1 \\ 1 & 1 & 1 & -1/R \end{pmatrix},$$

$$b_\sigma = U M_{LC}^{-1} \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ 0 \end{pmatrix}$$

where U is the input voltage. We take the following values in the per unit system: $U = 100$, $L = 10.10^{-3}$, $M = 9.9.10^{-3}$, $r = 100.10^{-3}$, $R = 1$, $C = 300.10^{-6}$, $\tau = 1/60000$.

Direct Method

For $S = [4,7] \times [4,7] \times [4,7] \times [15,17]$, $\tau = 1/60000$, we can synthesize the maximal controlled invariant subset $S' \subset S$, using Algorithm 6. A trajectory of the system starting at $x_0 = (5,5,5,16) \in S'$ is presented in Figure 7.18. The figure shows that all the trajectory lie inside S.

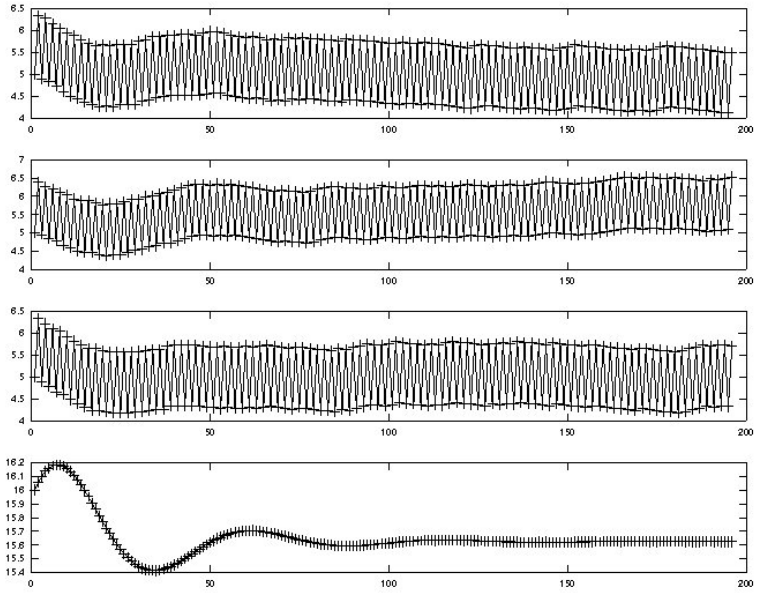


Figure 7.18: Discrete-time trajectory of 3-cells converter starting at $x_0 = (5,5,5,16)$ in $S = [4,7] \times [4,7] \times [4,7] \times [15,17]$ using the direct control method (from top to bottom: x_1, x_2, x_3, x_4 in function of time)

Indirect Method

For the safety region, we consider $S = [5.3,5.9] \times [5.3,5.9] \times [5.3,5.9] \times [15.5,16.5]$. It can be seen that the system is locally contractive in S with a contraction factor $\beta = 0.99202$. For the state space discretization, we take $\eta = 1/5$. This corresponds to a precision $\varepsilon = \eta/(1-\beta) \approx 21.6$. The abstract safety controller $\mathscr{A}_\eta$ associated to box S_η corresponds to a graph with several hundreds of vertices. A small part of the graph is given in Figure 7.19. This figure has been computed by a tool of ours implementing the Indirect Method. For example, there is a cycle passing through vertices numbered: 290, 311, 332, 353, 332, 311, 290. This corresponds to the application of the cyclic sequence of modes: $(4 \cdot 4 \cdot 4 \cdot 1 \cdot 2 \cdot 1)^*$.

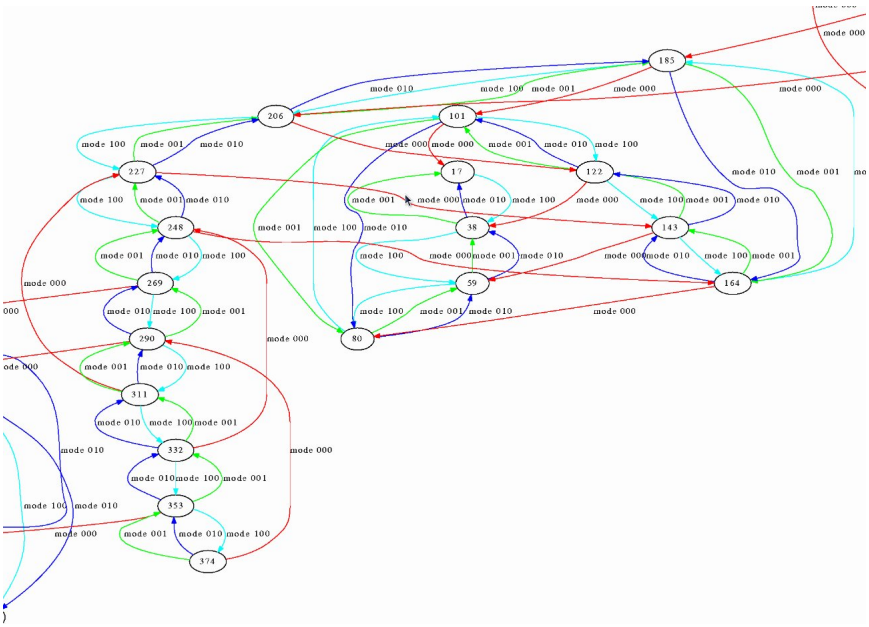


Figure 7.19: Partial graph of an abstract safety controller of the 3-cells boost converter, with $\eta = \frac{1}{5}$

For this control, the trajectory starting at point $x_0 = (5.4, 5.4, 5.4, 16)$, is given in Figure 7.20. We can see that the system does not stay inside the initial box S. However we can check that the system stays within the ε-over-approximation $\mathscr{B}(S, \varepsilon)$ of S with $\varepsilon = 21.6$. The value of ε is much too gross to give an interesting guarantee of safety. A finer precision ε would require a much smaller η and accordingly $\mathscr{A}_\eta$ with an overwhelming number of vertices. This tends to indicate that the indirect method (at least applied without further refinement) leads to prohibitively expensive computations for this example.

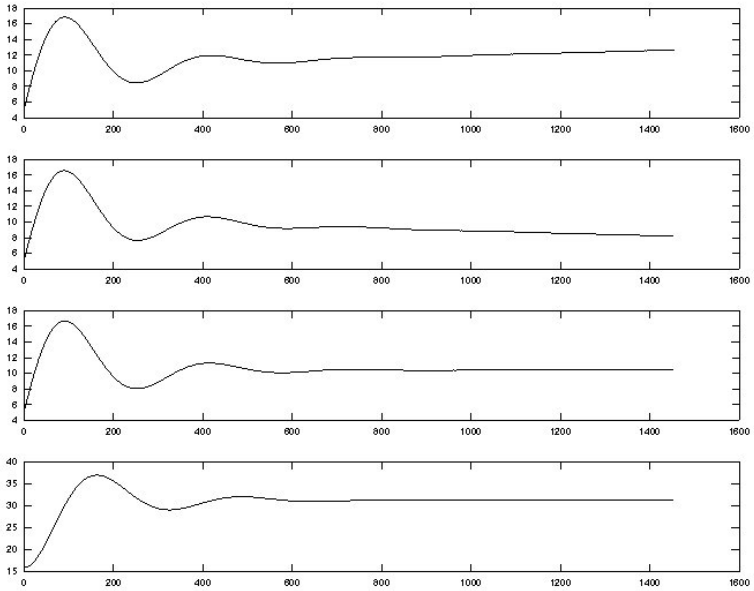


Figure 7.20: Trajectory of the 3-cells converter starting at $x_0 = (5.4, 5.4, 5.4, 16)$ with switching rule $(4 \cdot 4 \cdot 4 \cdot 1 \cdot 21)^*$ found by the indirect method (precision $\varepsilon = 21.6$)

Chapter 8

Stability Controllers

As explained in Chapter 7.5, the computation of maximal controlled invariant sets, found in the literature, relies essentially on a *backward* approach, computing iteratively the predecessors reachable sets. This is in keeping with the seminal work of Ramadge and Wonham for finite discrete systems [RW89]. Unfortunately, as pointed out in [Mit07], backward reachability constructs are more likely to suffer from *numerical stability* problems in systems displaying significant *contraction*, while contraction is generally a desirable and rewarding property of dynamical systems. In this chapter, we propose a forward method that allows to benefit from the contraction properties of our studied systems.

In this chapter, we are interested in the problem of practical stabilization: given a region R, find a switching rule that makes the system converge to a region located inside R. Such a switching rule corresponds to a *stability controller*. It is interesting to confine the trajectories in a region as small as possible. The problem is closely related to the problem of finding a controlled invariant subset of R as small as possible, ideally *minimal*. We present a direct forward-oriented method that *decomposes* a given state region R, and induces a state-dependent control that make the trajectories of the system converge to finite sets of points that, under certain conditions, correspond to *limit cycles*. The method can also be used for synthesizing *safety controllers* in order to prove safety properties.

Contributions Our contributions in this chapter are: the Decomposition Procedure, its enhancement and the theorems of Section 8.3, the theorem on limit cycles of Section 8.4, the implementation of our method in a tool discussed in Section 8.5, and the extension of the method to other frameworks of Section 8.6.

Outline of this chapter After explaining our motivation (Section 8.1), we first give some formal preliminaries (Section 8.2). We then present the decompo-

sition method (Section 8.3): The basic procedure is given in Section 8.3.1, its enhancement for proving safety properties in Section 8.3.3. We then apply the method to the synthesis of finite controlled invariants and limit cycles that attract the trajectories of the controlled system (Section 8.4). Some indications on the implementation of the decomposition procedure are given in Section 8.5. We show how an iteration of the decomposition procedure can be used to synthesize *reachability* controllers (Section 8.6.1). We suggest to exploit the *sensitivity* of the limit cycles in order to infer the values of physical parameters of the system using an inverse approach (Section 8.6.2). We explain how to extend the decomposition procedure in order to synthesize *robust safety* controllers in presence of disturbance (Section 8.6.3). We indicate how to extend the method for nonlinear systems in Section 8.6.4.

8.1 Motivation

Let us consider Figure 7.10 again. The set $S = [3.0, 3.4] \times [1.5, 1.8]$ is partitioned into a maximal controlled subset $S^* = P_1 \cup P_2$, made of two polyhedra P_1 and P_2, and an "uncontrollable" part colored in black made of two parts: a lower left part, say Q_1, and an upper right part, say Q_2, of S. Each polyhedron P_1, P_2, Q_1, Q_2 contains one (and only one) corner of S. The corner of S belonging to P_1 (resp. P_2) is controllable: if one applies mode 1 (resp. 2) to this corner, one finds a point belonging to $P_1 \cup P_2 \subset S$. In contrast the corners of S belonging to Q_1 and Q_2 are not controllable: the application of either mode 1 or mode 2 maps these corners in points located outside S. Now, let us ask the question: does there exist a *k-pattern*, i.e., a sequence of modes of length (at most) k, mapping the corners to points located *inside S*? If such patterns exist, we say that the corners are "k-controllable". In this example, one can see that it is the case for $k = 5$. Besides, if we divide S into 4 sub-boxes of equal size, say $V_1, \ldots, V_4$, each containing a corner, say $C_1, \ldots, C_4$, of S, one can see that the pattern, say π_i, that maps C_i inside S also maps the whole sub-box V_i inside S ($1 \leq i \leq 4$). This suggests a procedure of *decomposition* which splits S by bisection into sub-boxes, and looks for patterns which map the sub-boxes inside S. If this succeeds, we say that S is "k-controllable", or is a "controlled k-invariant set". The decomposition induces a *state-dependent control* strategy that makes any point of S return to S after at most k steps. In the following, we formalize these ideas.

8.2 Preliminaries

Definition 38. *Given a set $R \subset \mathbb{R}^n$ and a set $\{(V_i, \pi_i)\}_{i \in I}$ where I is a finite set of indices, V_i is a subset of $\mathbb{R}^n$ (for all $i \in I$), π_i is a k-pattern (for all $i \in I$), we say that $\Delta = \{(V_i, \pi_i)\}_{i \in I}$ is a* k-invariant decomposition of R *if:*

- $R = \bigcup_{i \in I} V_i$, *and*
- V_i *is R-invariant via π_i (i.e., $Post_{\pi_i}(V_i) \subset R$), for all $i \in I$.*

In the rest of this chapter, we will suppose that R is a *box* (i.e., a rectangular region of $\mathbb{R}^n$). Such a set R will be referred to as a *global (control) box*. The subsets V_is (with $i \in I$) will be themselves boxes included into R. They will be referred to as *local (control) boxes*.

Given a box $R \subset \mathbb{R}^n$ and a set Δ of the form $\{(V_i, \pi_i)\}_{i \in I}$ with $\bigcup_{i \in I} V_i = R$, we define $Post_\Delta$ as follows:

$$Post_\Delta(X) = \bigcup_{i \in I} Post_{\pi_i}(X \cap V_i), \quad \text{for all } X \subset R.$$

It is easy to show:

Proposition 9. *Given a box $R \subset \mathbb{R}^n$ and a set $\Delta : \{(V_i, \pi_i)\}_{i \in I}$ where the π_is ($i \in I$) are k-patterns, and with $\bigcup_{i \in I} V_i = R$, we have:*

- Δ *is a k-invariant decomposition of R iff $Post_\Delta(R) \subset R$.*
- *The image of a (compact) convex set by $Post_\Delta$ is a finite set of (compact) convex sets.*

NB: For the sake of simplicity, we will use $Post_\Delta(x)$ instead of $Post_\Delta(\{x\})$, when x is a point of $\mathbb{R}^n$.

Example 17. *(***Boost DC-DC Converter***). Let us consider Example 12 (see Chapter 7.4). In the case of the Boost DC-DC converter, one can show that, for $R = [1.55, 2.15] \times [1.0, 1.4]$, there is a decomposition $\Delta = \{(V_i, \pi_i)\}_{i=1,\ldots,4}$ with $V_1 = [1.55, 1.85] \times [1.0, 1.2]$, $V_2 = [1.85, 2.15] \times [1.0, 1.2]$, $V_3 = [1.85, 2.15] \times [1.2, 1.4]$, $V_4 = [1.55, 1.85] \times [1.2, 1.4]$, and $\pi_1 = (1 \cdot 1 \cdot 2 \cdot 2 \cdot 2)$, $\pi_2 = (2)$, $\pi_3 = (2 \cdot 1 \cdot 2)$, $\pi_4 = (1)$. One can check indeed that, for all $1 \leq i \leq 4$, $Post_{\pi_i}(V_i) \subset R$. This is visualized on Figure 8.1. In Section 8.3, we will explain how to generate such a decomposition.*

Definition 39. *Given a pattern π of the form $(u_1 \cdots u_m)$, and a set X, the* unfolding of X via π, *denoted by $Unf_\pi(X)$, is the set $\bigcup_{i=0}^{m} X_i$ with:*

- $X_0 = X$,

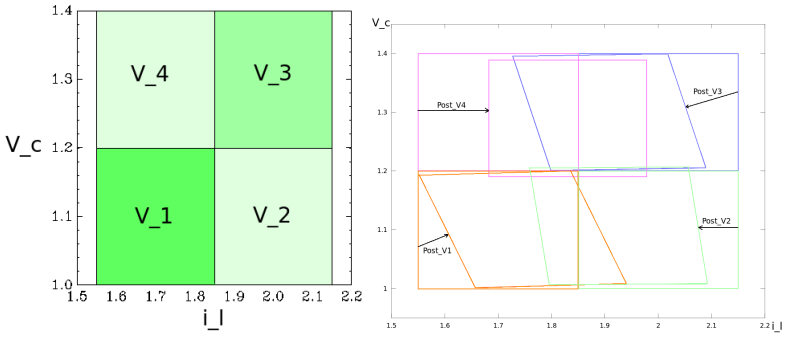


Figure 8.1: Decomposition Δ of $R = [1.55, 2.15] \times [1.0, 1.4]$ for the Boost DC-DC converter example (left), and visualization of $Post_\Delta(V_i) \subset R$, $i = 1, \ldots, 4$ (right)

- $X_{i+1} = Post_{u_{i+1}}(X_i)$, *for all* $0 \leq i \leq m-1$.

Definition 40. *Consider a k-invariant box R of decomposition* $\Delta = \{(V_i, \pi_i)\}_{i \in I}$. *The* Δ-unfolding *of R, denoted by* $Unf_\Delta(R)$, *is the set:*

$$\bigcup_{i \in I} Unf_{\pi_i}(V_i).$$

Example 18. *Figure 8.2 depicts the unfolding of R for the decomposition* Δ *of example 17, where dark gray (resp. light gray) indicates that mode 1 (resp. 2) applies.*

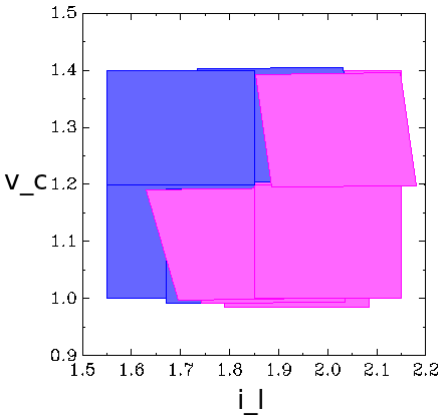


Figure 8.2: Δ-unfolding of $R = [1.55, 2.15] \times [1.0, 1.4]$ in the Boost DC-DC converter example where dark gray (resp. light gray) indicates that mode 1 (resp. 2) applies

Proposition 10. *Suppose that a box R has a k-invariant decomposition* Δ. *Then the* Δ*-unfolding of R is controlled invariant.*

Proof. Let us explain how such a control can be refined in order to make R', the Δ-unfolding of R, *controlled invariant*. We extend $\Delta : \{(V_i, \pi_i)\}_{i\in I}$ as follows. Each element of Δ is of the form (V, π) where π is of the form $(u^1 u^2 \cdots u^m)$. Such an element is replaced by m couples (V^1, u^1), (V^2, u^2), ..., (V^m, u^m) with $V^1 = V$, $V^2 = Post_{u^1}(V^1)$,..., $V^m = Post_{u^{m-1}}(V^{m-1})$. The decomposition Δ becomes a decomposition Δ' of the form $\{(V_i^j, u_i^j)\}_{i,j}$ with $\bigcup_{i,j} V_i^j = R'$ and $Post_{u_i^j}(V_i^j) \subset R'$ for all i, j. Hence, for each $x \in R'$, x belongs to some V_i^j, and $Post_{u_i^j}(x) \subset R'$. This shows that R' is controlled invariant. □

Control induced by the decomposition

The decomposition Δ induces a state-dependent control that makes any trajectory starting fom R go back to R within at most k steps: given a starting state x_0 in R, we know that $x_0 \in V_i$ for some $i \in I$ (since $R = \bigcup_{i\in I} V_i$); one thus applies π_i to x_0, which gives a new state x_1 that belongs to R (since V_i is R-invariant via π_i); the process is repeated on x_1, and so on iteratively. Given a point $x \in R$, we will denote by $succ_\Delta(x)$ the point of R obtained by applying π_i to x when x is in V_i. Note that a nondeterministic choice has to be done when a point x belongs to more than one local box V_i. We will suppose that we have an implicit selection function that operates a nondeterministic choice in such a case (for example, one can select the set V_i of least index containing x). When x belongs to a single local box V_i, then $succ_\Delta(x) = Post_\Delta(x)$.

A sequence of points $\{x_i\}_{i\geq 0}$, with $x_{i+1} = succ_\Delta(x_i)$ for all $i \geq 0$, is called a *discrete trajectory induced by* Δ, or more simply, a Δ*-trajectory*. [1]

We will also consider the *unfolding* of a Δ-trajectory, which corresponds to consider not only the successors of points via patterns, but also all the intermediate points generated by intermediate application of the modes forming the patterns. In the figures, for the sake of clarity, the points of Δ-trajectories will be linked together using straight lines, and similarly for their unfoldings.

Example 19. *A Δ-trajectory starting from the left upper corner of $R = [1.55, 2.15] \times [1.0, 1.4]$ for the Boost example, is presented in Figure 8.3 together with its unfolding.*

Using Proposition 10, one can prove safety properties of the controlled system, by showing $Unf_\Delta(R) \subset S$, where S is known to be a set of safe positions (see Section 8.3.3).

[1] We will sometimes denote such a trajectory under the form: $x_0 \rightarrow_{\pi_{i_1}} x_1 \rightarrow_{\pi_{i_2}} \cdots$ with $i_1, i_2, \cdots \in I$.

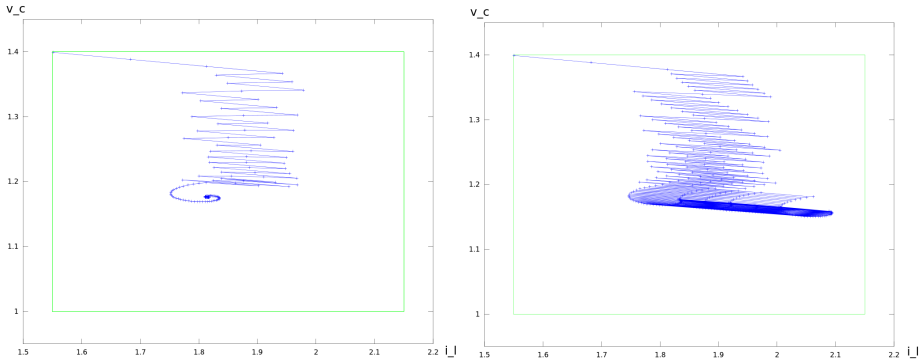


Figure 8.3: Δ-trajectory for the Boost example (left), and its unfolding (right)

8.3 Decomposition Procedure

8.3.1 Basic procedure

We suppose that we are given a global box $R \subset \mathbb{R}^n$. We now give a decomposition procedure which generates a k-invariant decomposition of R, as follows.

It first calls sub-procedure Find_Pattern in order to get a k-pattern such that R is R-invariant. If it succeeds, then it is done. Otherwise, it divides R into 2^n sub-boxes $V_1, \ldots, V_{2^n}$ of equal size. If for each V_i, Find_Pattern gets a k-pattern making it R-invariant, it is done. If, for some V_j, no such pattern exists, the procedure is recursively applied to V_j. It ends with success when a k-invariant decomposition of R is found, or failure when the maximal degree d of decomposition is reached.

Remark 2. *Since the local boxes V_is are cartesian product of closed intervals, two adjacent boxes V_i and V_j share a common facet.*

The algorithmic form of the procedure is given in Algorithms 7 and 8. (For the sake of simplicity, we consider the case of dimension $n = 2$, but the extension to $n > 2$ is straightforward.) The main procedure Decomposition(W,R,D,K) is called with R as input value for W, d for input value for D, and k as input value for K; it returns either $\langle \{(V_i, \pi_i)\}_i, True \rangle$ with $\bigcup_i V_i = W$ and $\bigcup_i Post_{\pi_i}(V_i) \subset R$, or $\langle \_, False \rangle$. Procedure Find_Pattern(W,R,K) looks for a K-pattern for which W is R-invariant (i.e., $Post_\pi(W) \subset R$): it selects all the K-patterns by non-decreasing length order until either it finds such a pattern π (output: $\langle \pi, True \rangle$), or no such pattern exists (output: $\langle \_, False \rangle$).

Remark 3. *Since R is a box, the inclusion test $Post_\pi(W) \subset R$ in procedure Find_Pattern(W,R,K) is implemented under the equivalent form $\square(Post_\pi(W)) \subset R$, which can be done in quadratic time (cf. Section 7.4.3).*

The correctness of the procedure is stated as follows.

Theorem 11. *If Decomposition(R,R,d,k) returns* $\langle \Delta, True \rangle$*, then* Δ *is a k-invariant decomposition of R. (Hence,* $Unf_{\Delta}(R)$ *is controlled invariant.)*

Very grossly, the complexity of procedure Find_Pattern is $O(n^3 N^k)$ since there are N^k patterns of length k, and the complexity of computation of successor states and inclusion test using zonotopes can be done in $O(kn^3)$, using zonotopes. This procedure is called by procedure Decomposition at most $2^{n \cdot d}$ times which corresponds to the number of sub-boxes in the cas of a maximal decomposition of length d. The worst complexity of the procedure is thus in $O(2^{n \cdot d} N^k)$. Unsurprisingly, it suffers from the *curse of dimensionality* regarding not only the state dimension n, but also the depth of decomposition d and the length of patterns k.

The examples treated with a simple implementation of the procedure scales up to 7 continuous variables. (see Section 8.5).

Note that there are boxes R for which Decomposition(R,R,d,k) does not succeed for any k and d. More generally, there are boxes R which are never k-invariant, for any k.

Algorithm 7: Decomposition(W,R,D,K)

Input: A box W, a box R, a degree D of decomposition, a length K of pattern

Output: $\langle \{(V_i, \pi_i)\}_i, True \rangle$ with $\bigcup_i V_i = W$ and $\bigcup_i Post_{\pi_i}(V_i) \subset R$, or $\langle \_, False \rangle$

```
1  (π, b) := Find_Pattern(W, R, K)
2  if b = True then
3      return ⟨{(W, π)}, True⟩
4  else
5      if D = 0 then
6          return ⟨_, False⟩
7      else
8          Divide equally W into (W1, W2, W3, W4)   /* (case n = 2)      */
9          (Δ1, b1) := Decomposition(W1,R,D − 1,K)
10         (Δ2, b2) := Decomposition(W2,R,D − 1,K)
11         (Δ3, b3) := Decomposition(W3,R,D − 1,K)
12         (Δ4, b4) := Decomposition(W4,R,D − 1,K)
13         return (Δ1 ∩ Δ2 ∩ Δ3 ∩ Δ4, b1 ∧ b2 ∧ b3 ∧ b4)
```

We present now a sufficient condition on the position of R for ensuring its k-invariance for some integer k.

Algorithm 8: Find_Pattern(W,R,K)

Input: A box W, a box R, a length K of pattern
Output: $\langle \pi, True \rangle$ with $Post_\pi(W) \subset R$, or $\langle \_, False \rangle$ when no pattern maps W into R

```
for i = 1...K do
    Π := set of patterns of length i
    while Π is non empty do
        Select π in Π
        Π := Π \ {π}
        if Post_π(W) ⊂ R then
            return ⟨π, True⟩
return ⟨_, False⟩
```

8.3.2 Sufficient Condition of Decomposition

Given a zone R, an invariant decomposition does not always exist. We give hereafter some geometrical conditions on the position of R that guarantee the decomposability of R when the system is contractive. For the sake of simplicity, we suppose that the switched system has only $|U| = 2$ modes, and the state space dimension is $n = 2$, but the reasoning extends to larger values of $|U|$ and n. We assume that matrix A_u associated with mode u ($u = 1,2$) is invertible and its eigenvalues have negative real parts. This implies that both modes are *contractive*. Let $e_u = -A_u^{-1} b_u$ be the unique attractive equilibrium point associated with mode u ($u = 1,2$). Let us define the "pure" switching rule $\mathscr{S}_u$ ($u = 1,2$) which applies repeatedly mode u to any point $x \in \mathbb{R}^2$. Let $\mathscr{C}_1$ (resp. $\mathscr{C}_2$) be the τ-sampled trajectory issued from e_1 (resp. e_2) under $\mathscr{S}_2$ (resp. $\mathscr{S}_1$) (i.e., $\mathscr{C}_1 = Post_2^*(e_1)$ and $\mathscr{C}_2 = Post_1^*(e_2)$). Since each mode is contractive, and e_u is the unique equilibrium point associated with mode u, any trajectory under control $\mathscr{S}_u$ ends to the equilibrium point e_u ($u = 1,2$), whatever the starting point of $\mathbb{R}^2$. In particular, $\mathscr{C}_1$ ends to e_2 and $\mathscr{C}_2$ to e_1, as depicted in Figure 8.4 for the boost converter (see examples 12 and 17).

Theorem 12. *Let Σ be a sampled switched affine system as defined above. Suppose that the reference point O is in $\mathscr{C}_1 \cup \mathscr{C}_2$. If $R \subset \mathbb{R}^2$ is a box whose interior contains O, then there exists a positive integer k such that R is k-invariant.*

Proof. Suppose $O \in \mathscr{C}_2$. (The case $O \in \mathscr{C}_1$ is symmetrical.) Consider a box R of interior $\dot{R}$ with $O \in \dot{R}$. There exists $\Delta_O > 0$ such that $\mathscr{B}(O, \Delta_O) \subset R$. Since $O \in \mathscr{C}_2$, we have:
(a) $e_2 \rightarrow_{\pi_1} O$, for some pattern $\pi_1 \in (1)^*$.

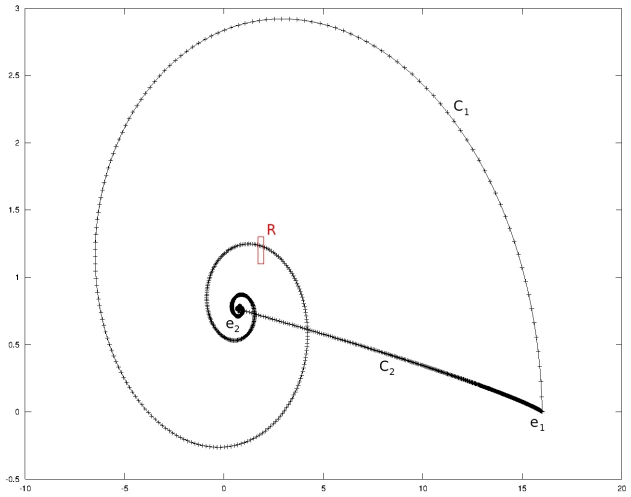


Figure 8.4: Trajectories $\mathscr{C}_1$, and $\mathscr{C}_2$ and zone $R = [1.7, 2] \times [1.1, 1.2]$ for the DC-DC converter example

Furthermore, for all $x \in R$, we have:
(b) $x \rightarrow_{\pi_2} x_1$ for some $x_1 \in \mathscr{B}(e_2, \Delta_O)$ and some pattern $\pi_2 \in (2)^*$, because e_2 is an attractive equilibrium point;
(c) $x_1 \rightarrow_{\pi_1} x_2$ for some $x_2 \in \mathscr{B}(O, \Delta_O)$, because of (a) and because mode 1 is contractive. This is depicted in Figure 8.5. It follows from (b)-(c) that, for all

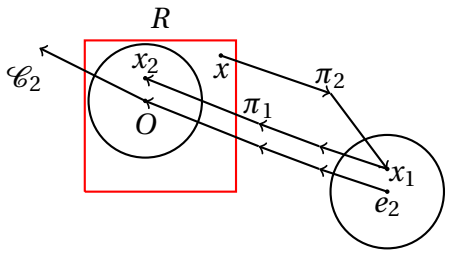


Figure 8.5: Illustration of the proof

$x \in R$: $x \rightarrow_{\pi_x} x_2$ for some $x_2 \in \mathscr{B}(O, \Delta_O)$ and some pattern $\pi_x \in (2^* 1^*)$. Hence, we have: $\mathscr{B}(x_2, \Delta_x) \subset R$ for some $\Delta_x > 0$. Since, for any π_x, $Post_{\pi_x}$ is continuous, $Pre_{\pi_x}(\mathscr{B}(x_2, \Delta_x))$ is an open subset of $\mathbb{R}^2$ containing x. Since R is a compact of $\mathbb{R}^2$, from the set $C = \{Pre_{\pi_x}(\mathscr{B}(Post_{\pi_x}(x), \Delta_x))\}_{x \in R}$, one can extract by Heine-Borel's theorem[2] a subset $C' = \{Pre_{\pi_{x_i}}(\mathscr{B}(Post_{\pi_{x_i}}(x_i), \Delta_{x_i}))\}_{i \in I}$, for some finite set of indices I, such that C' covers R and $\mathscr{B}(x_i, \Delta_{x_i})$ is R-invariant

[2]This theorem states that, if S is compact subset of $\mathbb{R}^n$, then for every open cover of S has a finite subcover

via π_i. This means that $C' \cap R$ is a k-invariant decomposition of R of the form $\{(V_i, \pi_i)\}_{i=1,\ldots,m}$, where m is the cardinal of I, k the maximum length of $\pi_1, \ldots, \pi_m$, and $V_i = \mathscr{B}(x_i, \Delta_{x_i}) \cap R$ is such that $\bigcup_{i=1}^{m} V_i = R$ and V_i is R-invariant via π_i $(1 \leq i \leq m)$. □

Theorem 12 gives an interesting locality condition on O (location on one of the "pure" trajectories linking the equilibrium points), for ensuring the existence of k-invariant boxes R. This justifies *a posteriori* the existence of a decomposition for the zone R of the DC-DC converter example 17 since it overlaps trajectory $\mathscr{C}_1$. Note also that R can be arbitrarily small as far as it intersects $\mathscr{C}_1$ (or $\mathscr{C}_2$).

8.3.3 Enhancement for Safety

Let us now explain how to extend the decomposition procedure in order to show additionally that we have $Unf_\Delta(R) \subset S$, where S is a given safety set containing R. This is done by adding an extra-input argument to procedures Decomposition and Find_Pattern corresponding to S. We then replace line 1 of Decomposition(W,R,D,K,S) by Find_Pattern(W,R,K,S), and line 6 of procedure Find_Pattern by:

If $Post_\pi(W) \subset R$ ***And*** $Unf_\pi(W) \subset S$ **then**

In other words: if π is a pattern of the form $(u_1 \cdots u_m)$ with $u_1, ..., u_m \in U$, we check additionally $W_i \subset S$ for all $1 \leq i \leq m$, where the W_is are the intermediate sets defined by $W_1 = Post_{u_1}(W)$, ..., $W_m = Post_{u_m}(W_{m-1})$. In a straightforward manner, we have:

Theorem 13. *If the procedure Decomposition(R,R,d,k,S) returns* $\langle \Delta, True \rangle$, *then* $Unf_\Delta(R)$ *is controlled invariant. Furthermore, we have:* $Unf_\Delta(R) \subset S$.

Hence, the system under the control inferred by the procedure (when it succeeds), is guaranteed to be safe.

The enhanced procedure has been implemented (see Section 8.5 for details). We illustrate its application on the boost DC-DC converter of example 12.

Example 20. *For R, we now consider the box* $[1.75, 1.95] \times [1.14, 1.26]$, *which corresponds to a medium value* 1.85 *for* i_l *with* ± 0.1 *for variability, and medium value* 1.20 *for* v_c *with* ± 0.06 *for variability. For the safety region, we take* $S = [1.7, 2.0] \times [1.10, 1.30]$, *which corresponds to an additional variability of* ± 0.05 *for* i_l *and* ± 0.04 *for* v_c. *The application of algorithm 7 to R and S, with* $k = 10$ *and* $d = 4$ *succeeds, yielding a k-invariant decomposition* Δ *of the form*

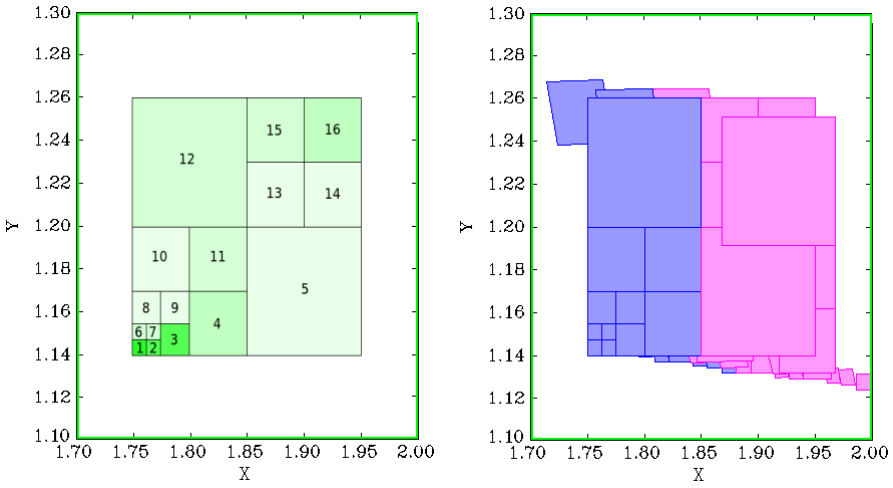


Figure 8.6: Left: decomposition Δ for boost converter of $R = [1.75, 1.95] \times [1.14, 1.26]$; right: Δ-unfolding where dark gray (resp. light gray) indicates mode 1 (resp. 2), with enclosing box $S = [1.7, 2.0] \times [1.1, 1.3]$

$\{(V_j, \pi_j)\}_{j=1,\ldots,16}$ *of* R [3] *satisfying* $Unf_\Delta(R) \subset S$. *The k-invariant decomposition* Δ *of R is depicted in the left part of Figure 8.6, and the* Δ*-unfolding is depicted in the right part, with two gray colors corresponding to the different control modes: dark gray (resp. light gray) indicates that mode 1 (resp. 2) should be applied. Figure 8.6 has been generated by our tool MINIMATOR. The unfolded* Δ*-trajectory of the system starting at point* $(1.75, 1.26)$*, is depicted in Figure 8.7 generated by a script of ours that run simulation given a controller.*

8.3.4 Applications of the Enhanced Decomposition Procedure

We now apply the decomposition procedure (enhanced for safety properties) to several examples. For each example, we are given a global control box R and a safety set S which contains it. We generate a k-invariant decomposition Δ of R satisfying $Unf_\Delta(R) \subset S$, thus proving that $Unf_\Delta(R)$ is a controlled invariant, and that the system is safe.

Example 21. *Let us consider the Helicopter motion example 14 of Chapter 7.4. We take* $R = [-0.3, 0.3] \times [-0.5, 0.5]$ *in plane* (x_1, x_2)*. This corresponds to an equilibrium zone centered at the state* $(0, 0)$ *of the ground vehicle, and a variability of* ± 0.3 *for position and* ± 0.5 *for velocity. We take* $S = [-0.4, 0.4] \times [-0.7, 0.7]$ *for the*

[3] The associated patterns are: $\pi_1 = (1122122122)$, $\pi_2 = (12121222)$, $\pi_3 = (12122122)$, $\pi_4 = (122)$, $\pi_5 = (2)$, $\pi_6 = (12)$, $\pi_7 = (12)$, $\pi_8 = (1)$, $\pi_9 = (1)$, $\pi_{10} = (1)$, $\pi_{11} = (12)$, $\pi_{12} = (12)$, $\pi_{13} = (2)$, $\pi_{14} = (2)$, $\pi_{15} = (12)$, $\pi_{16} = (221)$.

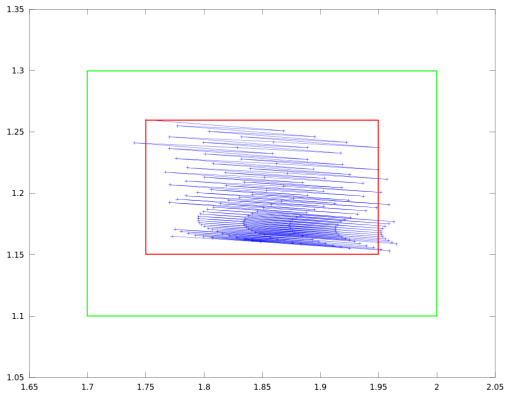


Figure 8.7: Unfolded Δ-trajectory of the boost converter starting at $(1.75, 1.26)$ (inner box $R = [1.75, 1.95] \times [1.14, 1.26]$, outer box $S = [1.7, 2.0] \times [1.1, 1.3]$)

safety region, which corresponds to an additional variability of ± 0.1 *for position and velocity. (This is the same safety zone S as in [DLHT11].)*

The application of algorithm 7 to R and S with $k = 6$ *and* $d = 4$ *succeeds, yielding a k-invariant decomposition* Δ *of R of the form* $\{(V_i, \pi_i)\}_{i=1,\ldots,10}$[4]

satisfying $Unf_\Delta(R) \subset S$. *The decomposition* Δ *is represented in Figure 8.8. The unfolding of R is depicted in Figure 8.9. These figures have been generated by our tool MINIMATOR. The unfolding is divided into regions of three different colors corresponding to the different control modes: color dark gray (resp. medium gray, light gray) indicates that mode* 10 *(resp.* -10, 0*) should be applied. The surrounding box is* $S = [-0.4, 0.4] \times [-0.6, 0.6]$.

The unfolded Δ*-trajectory, in plane* $(x, \dot{x})$, *of the system starting at point* $(-0.3, 0.5)$, *is depicted in Figure 8.10, generated by a script of ours that run simulation given a controller.*

Example 22. *(***Two-room building heating***) Let us consider the example of the Two-Room Building Thermal problem (see Example 13 of Chapter 7.4). For the safety zone, we take* $S = [20, 22] \times [20, 22]$, *as in [Gir12].*

The application of the algorithm 7 to R and S with $k = 4$ *and* $d = 2$ *succeeds, yielding a k-invariant decomposition* Δ *of the form* $\{(V_j, \pi_j)\}_{j=1,\ldots,10}$ *of* R[5] *satisfying* $Unf_\Delta(R) \subset S$. *The decomposition* Δ *is depicted in Figure 8.11, generated*

[4] The associated patterns are:
$\pi_1 = (-10 \cdot -10 \cdot -10 \cdot -10 \cdot -10 \cdot 0 \cdot 10)$, $\pi_2 = (-10)$, $\pi_3 = (0)$, $\pi_4 = (-10 \cdot -10 \cdot -10 \cdot 10)$, $\pi_5 = (-10)$, $\pi_6 = (0)$, $\pi_7 = (-10)$, $\pi_8 = (10 \cdot 10 \cdot 0 \cdot 0)$, $\pi_9 = (0)$, $\pi_{10} = (10 \cdot 10 \cdot 10 \cdot 10 \cdot 0 \cdot 10 \cdot -10)$.

[5] The associated patterns are: $\pi_1 = (1010)$, $\pi_2 = (1)$, $\pi_3 = (10)$, $\pi_4 = (0)$, $\pi_5 = (0)$, $\pi_6 = (1)$, $\pi_7 = (0)$, $\pi_8 = (0)$, $\pi_9 = (0)$, $\pi_{10} = (10)$.

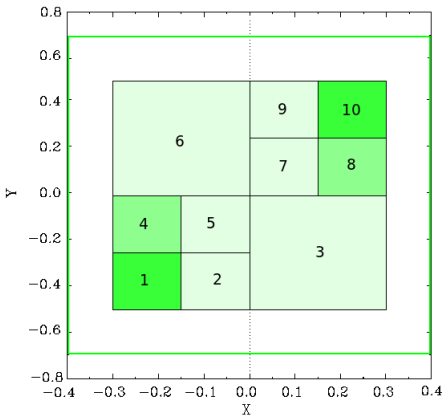


Figure 8.8: k-invariant decomposition for helicopter motion

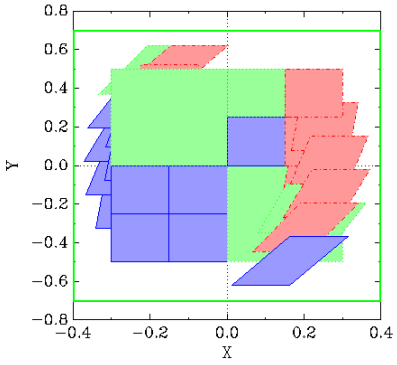


Figure 8.9: Δ-unfolding for helicopter motion where dark gray (resp. medium gray, light gray) indicates mode 10 (resp. −10, 0). (The enclosing box is S)

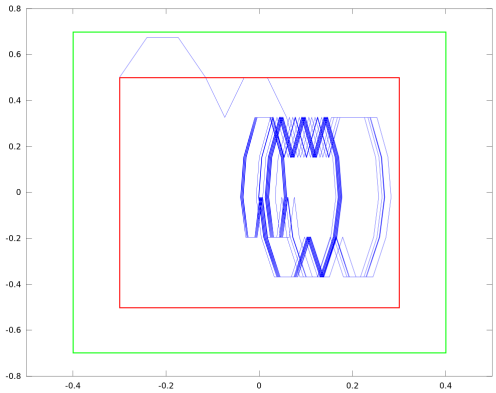


Figure 8.10: Unfolded Δ-trajectory of helicopter motion in plane $(x, \dot{x})$ starting at $(-0.3, 0.5)$ (inner box: R, outer box: S)

by MINIMATOR. The unfolding of R is represented on Figure 8.12, generated by MINIMATOR. The unfolding is divided into regions of two colors corresponding to the different control modes: the dark gray (resp. light gray) color indicates that control 0 (resp. 1) should be applied. The outer box is the safety zone S.

The controlled system has been simulated using Octave [Oct13]. A simulation is depicted in Figure 8.13, generated by a script of ours, for starting temperature point (20.25, 21.75).

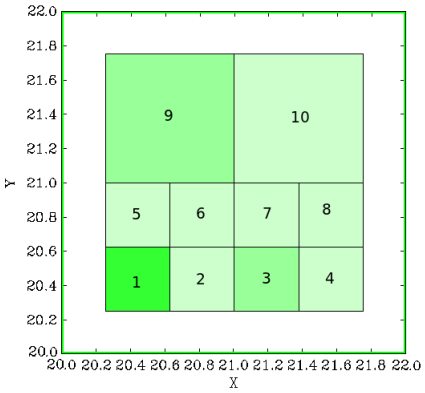


Figure 8.11: k-invariant decomposition for heating system

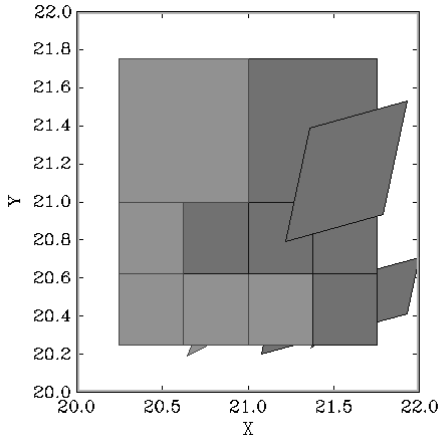


Figure 8.12: Δ-unfolding for heating system where dark gray (resp. light gray) indicates mode 0 (resp. 1). (outer box: $S = [20,22] \times [20,22]$)

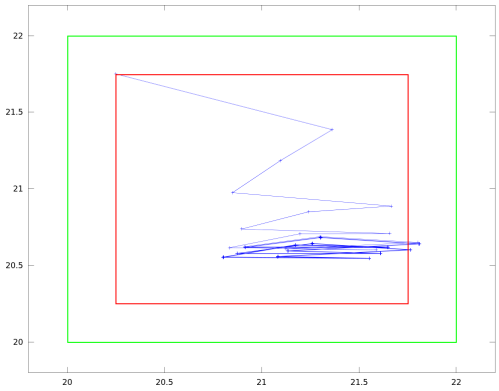


Figure 8.13: Unfolded Δ-trajectory of heating system in plane (T_1, T_2), starting at $(20.25, 21.75)$ (inner box $R = [20.25, 21.75] \times [20.25, 21.75]$, outer box $S = [20,22] \times [20,22]$])

8.4 Limit Cycles

In Figures 8.3, we see that the (unfolding of the) Δ trajectory seems to converge to a cycle. We now formally state that under certain assumptions, this is actually the case. We suppose that we are given a global box R and a decomposition $\Delta = \{(V_i, \pi_i)\}_{i \in I}$ produced by the decomposition algorithm of Section 8.3. We denote the union of the set of borders of V_i $(i \in I)$ by $\partial\Delta$. (Recall that two adja-

cent boxes V_i and V_j share a common border; see Remark 2.)

We show how to produce *attractors* of R, using an iteration of $Post_\Delta$. Furthermore, under certain assumptions which are often met in practice, these attractors are made of *finite* subsets of points corresponding to *limit cycles* which attract all the Δ-trajectories starting in R.

The idea is the following: since $Post_\Delta(R) \subset R$, one has $Post_\Delta^{i+1}(R) \subset Post_\Delta^i(R)$ for all $i \geq 0$, and the limit set $R_\Delta^* = \bigcap_{i\geq 0} Post_\Delta^i(R)$ is well defined and non-empty.

Lemma 13. *Consider a k-invariant decomposition $\Delta = \{(V_i, \pi_i)\}_{i\in I}$ of R. The sequence $\{R_\Delta^j\}_{j\geq 0}$ defined by:*

- $R_\Delta^0 = R$,
- $R_\Delta^{j+1} = Post_\Delta(R_\Delta^j)$

is a decreasing nested sequence and the set $R_\Delta^* = \bigcap_{j\geq 0} R_\Delta^j$ is well-defined. Furthermore, R_Δ^* is an attractor set *of R, i.e.:*

1. $Post_\Delta(R_\Delta^*) = R_\Delta^*$ *(*invariance *property)*
2. $\forall x \in R,\ d(Post_\Delta^j(x), R_\Delta^*) \to 0$ *as j tends to ∞*[6] *(*attraction *property).*

We now make the following assumptions:

(H1): All the modes are locally contractive in R.

(H2): There exists $N \in \mathbb{N}$ such that $Post_\Delta^N(R) \cap \partial\Delta = \emptyset$.

These assumptions will be discussed at Section 8.4.2. We show that under these assumptions, R_Δ^* is a *finite* set of points composed of disjoint "cycles". Furthermore, each Δ-trajectory starting from a point of R converges to one of these cycles. This is formally stated as follows.

Definition 41. *A* cycle *is a finite set of points of R of the form $\{y_0, y_1, \ldots, y_{m-1}\}$ with $y_0 \to_{\pi_{i_1}} y_1 \to_{\pi_{i_2}} \cdots \to_{\pi_{i_m}} y_m = y_0$ for some patterns $\pi_{i_1}, \ldots, \pi_{i_m}$ of Δ.*

Theorem 14. *Under assumptions (H1)-(H2), we have:*

1. *R_Δ^* is a finite union of (disjoint) cycles.*
2. *The Δ-unfolding of each cycle of R_Δ^* is a controlled invariant finite set.*

[6] $d(y, Z)$ denotes the smallest distance between a point $y \in \mathbb{R}^n$ and any point of $Z \subset \mathbb{R}^n$.

3. *Each* Δ*-trajectory* $\{x_0, x_1, \ldots\}$ converges to a cycle *of the form* $\{y_0, y_1, \ldots, y_{m-1}\}$ *in the following sense:*

$$\exists M \in \mathbb{N}\ \forall \ell = 0, \ldots, m-1 \ \lim_{i \to \infty} x_{M+i \cdot m+\ell} = y_\ell.$$

Note that, although the Δ-unfolding of each cycle is finite, it is not ensured to be a *minimal* controlled invariant: it is *a priori* possible that a strict subset of the Δ-unfolding be itself controlled invariant.

The next subsection present a detailed proof of Theorem 14.

8.4.1 Proof of the Convergence towards Limit Cycles

Consider a k-invariant decomposition $\Delta = \{(V_i, \pi_i)\}_{i \in I}$ of R, and the sequence $\{R^j_\Delta\}_{j \geq 0}$ and its limit R^*_Δ.

Let us consider the (compact) convex sets W^k_σ where $k \in \mathbb{N}$ and $\sigma \in I^k$, defined as follows:

- $W^0_\epsilon = R$ where ϵ denotes the empty sequence.
- $W^{k+1}_{(i \cdot \sigma)} = Post_{\pi_i}(W^k_\sigma \cap V_i)$ with $i \in I$ and $\sigma \in I^k$.

It is easy to show that, for all $k \in \mathbb{N}$ and all $\sigma \in I^k$, W^k_σ is a compact convex set such that
(1): $Post^k_\Delta(R) = \bigcup_{\sigma \in I^k} W^k_\sigma$.
It follows from assumption (H2) that, for all sequence $\sigma \in I^N$, $W^N_\sigma \cap \partial\Delta = \emptyset$, therefore $W^N_\sigma \subset \mathring{V}_i$ for some $i \in I$, and $Post_\Delta(W^N_\sigma) = Post_{\pi_i}(W^N_\sigma)$. Since $Post^{N+1}_\Delta(R) \subset Post^N_\Delta(R)$, $Post_\Delta(W^N_\sigma) = Post_{\pi_i}(W^N_\sigma)$ is a compact convex set included into $Post^N_\Delta(R)$, and is therefore included into one of the convex components of $Post^N_\Delta(R)$. We have
(2): $Post_\Delta(W^N_\sigma) \subset W^N_{\sigma'}$ for some $\sigma' \in I^N$.
Now, for all Δ-trajectory $\{x_0, x_1, \ldots\}$, we have
(3): $\forall k \geq N\ \exists \sigma \in I^N :\ x_k \in W^N_\sigma$
because, for all $k \geq N$, $x_k \in Post^k_\Delta(R) \subset Post^N_\Delta(R) = \bigcup_{\sigma \in I^N} W^N_\sigma$. By rewriting the elements of $\{W^N_\sigma\}_{\sigma \in I^N}$ under the form $\{W_1, \ldots, W_M\}$, and denoting the set $\{1, \ldots, M\}$ by J, we recapitulate these results as follows:

Proposition 11. *There exist M compact convex sets $W_1, \ldots, W_M$ with $\bigcup_{j=1}^M W_j \cap \partial\Delta = \emptyset$, such that*
(1′): $Post^N_\Delta(R) = \bigcup_{j=1}^M W_j$
(2′): $\forall j \in J\ \exists j' \in J :\ Post_\Delta(W_j) \subset W_{j'}$.
(3′): for all Δ-trajectory $\{x_0, x_1, \ldots\}$, $\forall i \geq N\ \exists j \in J :\ x_i \in W_j$.
The element j' associated with j in (2′) will be denoted by $s(j)$. (If there are more than one such a j', an arbitrary one is selected.)

Using part (2′) of Proposition 11, one can define a directed graph where J is the set of vertices, and there is an oriented edge from $j \in J$ to $j' \in J$ iff $j' = s(j)$ (i.e., $Post_\Delta(W_j) \subset W_{j'}$). The strongly connected components of this graph are cyclic (because each vertex of the graph has a unique outgoing edge).

In the following, we will denote a cyclic subgraph by $\mathscr{C} = (j_0, \ldots, j_{m-1})$ with $j_{\ell+1} = s(j_\ell)$ for $0 \leq \ell \leq m-1$, using the convention: $j_m = j_0$. For every element j of $\mathscr{C}$, we have $s^m(j) = j$, i.e.: $Post^m_\Delta(W_j) \subset W_j$. More generally, $Post^{(i+1)\cdot m}_\Delta(W_j) \subset Post^{i\cdot m}_\Delta(W_j)$. We can define a decreasing sequence of nonempty compact convex sets $Post^{i\cdot m}_\Delta W_j$ for all $i \geq 0$. The limit set $\bigcap_{i\geq 0} Post^{i\cdot m}_\Delta(W_j)$ is a nonempty compact set. Furthermore, since by (H1) all the modes are contractive, it is easy to see than this limit set is reduced to a point, that will be denoted by z_j. Every cycle of indices $\mathscr{C} = (j_0, \ldots, j_{m-1})$ thus corresponds to a cycle of points $Z_\mathscr{C} = (z_{j_0}, \ldots, z_{j_{m-1}})$. Furthermore, it is easy to show that, for all $0 \leq \ell \leq m-1$, we have $succ_\Delta(z_{j_\ell}) = z_{j_{\ell+1}}$. Let us denote the set of vertices of all the cyclic subgraphs by J'. An illustration of the form of such a graph is given in Figure 8.14.

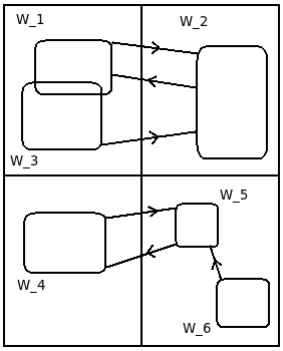


Figure 8.14: Illustration of the graph of W_js with $J = \{1,2,3,4,5,6\}$ and $J' = \{1,2,4,5\}$

These results are recapitulated in the following.

Proposition 12. *For all cycle $\mathscr{C} = (j_0, \ldots, j_{m-1})$ of J', we have:*
$\forall\, j \in \mathscr{C}:\ Post^m_\Delta(W_j) \subset W_j$.
Furthermore, for all $j \in \mathscr{C}$, the set $\bigcap_{i\geq 0} Post^{i\cdot m}_\Delta(W_j)$ is well defined and equal to a point denoted by z_j. We have, for all $\ell = 0, \ldots m-1$: $succ_\Delta(z_{j_\ell}) = z_{j_{\ell+1}}$.

Consider now the vertices of the graph that are $J \setminus J'$. Each of them is the destination of a finite number of acyclic paths. This means that after a finite number of iterations of $Post_\Delta$, no point will belong to W_j for $j \in J \setminus J'$[7].

[7] More formally, no point will belong to a W_j for $j \in J \setminus J'$ if it does not belong to a $W_{j'}$ for $j' \in J'$

Formally: $\exists M \geq N\ \forall j \in J \setminus J'\ Post^M_\Delta(R) \cap W_j = \emptyset$. Furthermore: $\forall k \geq M\ Post^k_\Delta(R) \subset \bigcup_{j\in J'} W_j$. It follows that, for all Δ-trajectory $\{x_0, x_1, \ldots\}$, x_M belong to W_j for some $j \in J'$. Let $\mathscr{C} = (j_0, \ldots, j_{m-1})$ be the cycle which contains j. Modulo a circular permutation of $\mathscr{C}$, one can suppose $j = j_0$. Then, for all $0 \leq \ell \leq m-1$, $x_{M+\ell}$ belongs to W_{j_ℓ}. More generally, $x_{M+i\cdot m+\ell} \in Post^{i\cdot m}(W_{j_\ell})$. It follows: $\lim_{i\to\infty} x_{M+i\cdot m+\ell} = \bigcap_{i\geq 0} Post^{i\cdot m}_\Delta(W_{j_\ell}) = z_{j_\ell}$. We have:

Theorem 15. *Every Δ-trajectory $\{x_0, x_1, \ldots\}$ converges to a limit cycle in the following sense: for all initial point $x_0 \in R$, there exists a cycle $\mathscr{C} = (j_0, \ldots, j_{m-1})$ of J', a cycle of points $Z_\mathscr{C} = (z_{j_0}, \ldots, z_{j_{m-1}})$ and an integer $M \in \mathbb{N}$ such that, for all $\ell = 0, \ldots, m-1$*

- $\forall i \geq 0 :\ x_{M+i\cdot m+\ell} \in Post^{i\cdot m}_\Delta(W_{j_\ell}) \subset W_{j_\ell}$.
- $\lim_{i\to\infty} x_{M+i\cdot m+\ell} = z_{j_\ell}$.

The expression $Post^m_\Delta(z_{j_\ell}) = z_{j_\ell}$ of Theorem 14 gives a practical method to compute z_{j_ℓ} ($\ell = 0, \ldots, m-1$). Indeed, we have that $z_{j_\ell} \in V_i$ for some $i \in I$. Let us denote such a i by $\phi(j_\ell)$, and let $\pi = (\pi_{\phi(j_0)} \cdots \pi_{\phi(j_{m-1})})$. Since $Post^m_\Delta(z_{j_0}) = Post_\pi(z_{j_0}) = C_\pi \cdot z_{j_0} + d_\pi$ for some matrix C_π and vector d_π, we can compute z_{j_0} as a solution of the equation $z_{j_0} = C_\pi \cdot z_{j_0} + d_\pi$. Similar equations hold for z_{j_ℓ} with $\ell = 1, \ldots, m-1$. Furthermore, we have:

Proposition 13. $R^*_\Delta = \{z_j \mid j \in J'\}$.

Proof. We know that there exists $K > 0$ such that, for all $k \geq K$, $Post^k_\Delta(R) = \bigcup_{j\in J'} Post^k_\Delta(W_j)$. We also know: $\bigcap_{k\geq 0} Post^k_\Delta(W_j) = \{z_j\}$. It follows $R^*_\Delta = \bigcap_{k\geq 0} Post^k_\Delta(R) = \bigcup_{j\in J'} \bigcap_{k\geq 0} Post^k_\Delta(W_j) = \{z_j \mid j \in J'\}$. □

The elements of $\{z_j \mid j \in J'\}$ are grouped together into cyclic sets of points. For all cyclic set of points $Z_\mathscr{C}$, we have $Post_\Delta(Z_\mathscr{C}) = Z_\mathscr{C}$. It follows:

Proposition 14. *For all cyclic set of points $Z_\mathscr{C}$, the Δ-unfolding of $Z_\mathscr{C}$ is a controlled invariant set.*

From Theorem 15 Propositions 13 and 14 it follows Theorem 14.

8.4.2 Discussing Assumptions (H1) and (H2)

Under assumptions (H1) and (H2), we have stated that each trajectory converges to a finite cyclic set of points.

The first assumption is classical in order to ensure convergence results, and was also used in Chapter 7.5. Actually, even if some of the modes are not contractive, we may observe the convergence to finite cyclic sets of points because

of the contractivity of the patterns involved by the control. This is the case in the helicopter motion example (see Figure 8.10). However, if none of the modes are (locally) contractive, then there is no limit cycle, and R^*_Δ is infinite as illustrated in the following example.

Example 23. *For this example, we use modes associated to repulsive homothetic transformation. There are* 4 *modes such that, close to each corner of the global box* $R = [-1,1] \times [-1,1]$*, there exists a fixpoint for one of the mode. We take for the dynamics of the modes:* $A_1 = A_2 = A_3 = A_4 = \begin{pmatrix} 1.5 & 0 \\ 0 & 1.5 \end{pmatrix}$, $b_1 = \begin{pmatrix} 0.6 \\ 0.6 \end{pmatrix}$, $b_2 = \begin{pmatrix} -0.6 \\ 0.6 \end{pmatrix}$, $b_3 = \begin{pmatrix} -0.6 \\ -0.6 \end{pmatrix}$, *and* $b_4 = \begin{pmatrix} 0.6 \\ -0.6 \end{pmatrix}$. *The* Δ*-decomposition is presented in Figure 8.15 computed by MINIMATOR. We have* $V_1 = [-1,0] \times [-1,0]$ *associated to pattern* $\pi_1 = (1)$, $V_2 = [0,1] \times [-1,0]$ *associated to pattern* $\pi_2 = (2)$, $V_3 = [0,1] \times [0,1]$ *associated to pattern* $\pi_3 = (3)$, *and* $V_4 = [-1,0] \times [0,1]$ *associated to pattern* $\pi_4 = (4)$. *A* Δ*-trajectory is depicted in Figure 8.16 simulated by a script of ours. One can see a chaotic behavior and an absence of convergence towards a limit cycle.*

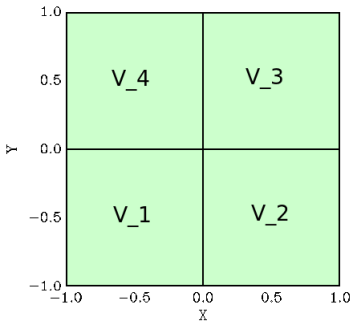


Figure 8.15: Decomposition Δ of R for the non contractive example

Assumption (H2) is justified as follows. When $Post_\Delta$ is applied to R for the first time, the local boxes are transformed into $|I|$ convex sets. If such a set, say W, crosses a border of $\partial\Delta$ and partly belongs to, say, two local boxes V_1 and V_2, it will be split into two sets $Post_{\pi_1}(W \cap V_1)$ and $Post_{\pi_2}(W \cap V_2)$ at the next application of $Post_\Delta$. The number of convex sets generated at each application of $Post_\Delta$ thus increases repeatedly until no image crosses a border, which happens at step N by assumption (H2). The images generated by further application of $Post_\Delta$ will then never cross $\partial\Delta$: these images will either disappear or shrink towards single points by assumption (H1). In the absence of assumption (H2), the number of connected sets of $Post^k_\Delta(R)$ can increase indefinitely. Assumption (H2) thus seems to be a necessary condition for the finiteness of R^*_Δ.

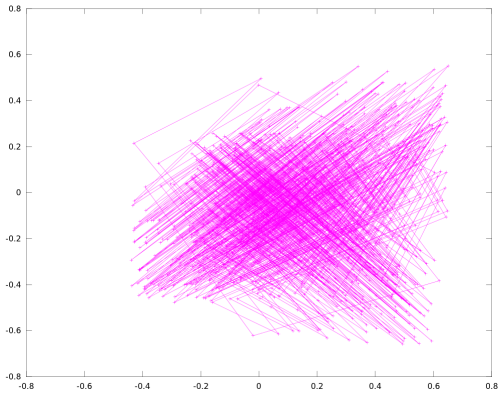


Figure 8.16: Δ-trajectory for the non contractive example

8.4.3 Illustrative Examples

We now illustrate the convergence of $Post^k_\Delta$ to a cyclic set of points as k tends to infinity, on the boost and two-tank example.

Example 24. *(**Boost DC-DC Converter**). One has already seen that the modes of the Boost converter are locally contractive (see Example16), hence (H1) is satisfied. Likewise, (H2) is satisfied: for $N = 100$, $Post^N_\Delta(R)$ is entirely contained into the local box V_1 of the decomposition Δ (see Figure 8.17 depicting the iterated images $Post^k_\Delta$ for $k = 0, 20, 40, 60, 80, 100$). These figures have been generated by MINIMATOR. The limit set R^*_Δ is here composed of a unique limit cycle that is*

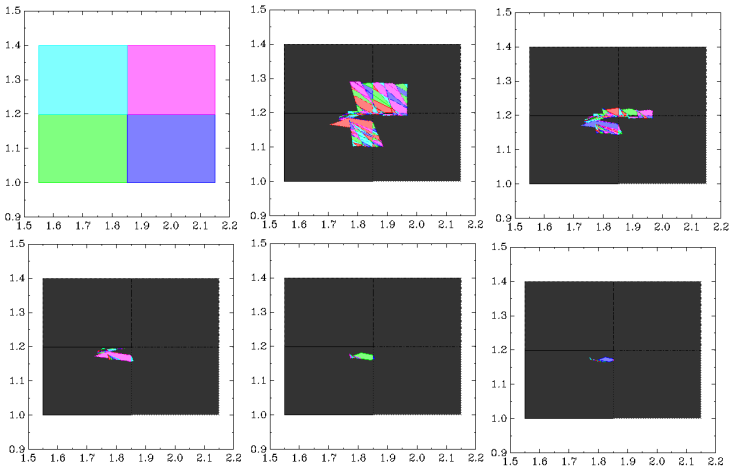


Figure 8.17: Visualization of $Post^k_\Delta$ for $k = 0, 20, 40, 60, 80, 100$

made of a single point $y_0 \in V_1$. We have: $y_0 \rightarrow_{\pi_1} y_1 = y_0$, with $\pi_1 = (1 \cdot 1 \cdot 2 \cdot 2 \cdot 2)$.

The Δ-unfolding of this limit cycle is thus made of 5 points corresponding to the composing modes of π_1 (see Figure 8.18, generated by a script of ours).

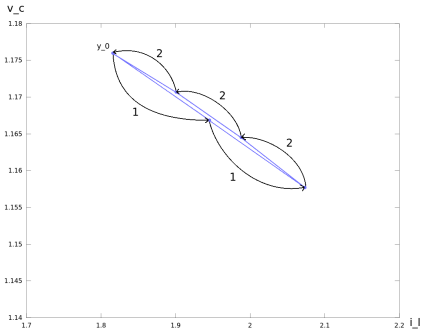


Figure 8.18: Δ-unfolding of the limit cycle $\{y_0\}$ for the Boost example

Example 25. *(**Two-Tank**). The two-tank system example is taken from [His01]. The system consists of two tanks and two valves. The first valve adds to the inflow of tank 1, the second valve is a drain valve for tank 2. There is also a constant outflow from tank 2 caused by a pump. The system is linearized at a desired operating point. The objective is to keep the water level in both tanks within limits using a discrete open/close switching strategy for the valves. Let the water level of tank 1 and 2 be given by x_1 and x_2 respectively. The behavior of x_1 is given by: $\dot{x}_1 = -x_1 - 2$ when the tank 1 valve is closed, and $\dot{x}_1 = -x_1 + 3$ when it is closed. Likewise, x_2 is driven by: $\dot{x}_2 = x_1$ when the tank 2 valve is closed and $\dot{x}_2 = x_1 - x_2 - 5$ when it is closed. Using $R = [-1.5, 2.5] \times [-0.5, 1.5]$ as a control box, we obtain the decomposition depicted in Figure 8.19, generated by MINIMATOR. With $V_1 = [-1.5, 0.5] \times [-0.5, 0.5]$ associated to pattern $\pi_1 = (2 \cdot 3 \cdot 3)$, $V_2 = [0.5, 2.5] \times [-0.5, 0.5]$ to $\pi_2 = (2)$, $V_3 = [0.5, 2.5] \times [0.5, 1.5]$ to $\pi_3 = (1 \cdot 4)$ and $V_4 = [-1.5, 0.5] \times [0.5, 1.5]$ to $\pi_4 = 3$. Figure 8.20 depicts a discrete trajectory of the two tank system, and its Δ-unfolding. These figures have been obtained by our script generating simulation of trajectories under a controller.*

One can check that all the modes of the two-tank system are contractive (the eigenvalues of all the associated matrix have negative real part). Hence (H1) is satisfied. Likewise, (H2) is satisfied: for $N = 10$, $Post^N_\Delta(R)$ does not intersect the borders of Δ (see Figure 8.21 depicting the iterated images $Post^k_\Delta(R)$, for $k = 0, 5, 10, 15, 20, 25$ computed by our tool MINIMATOR).

The limit set R^*_Δ is here composed of a unique limit cycle of the form $\{y_0, y_1, y_2, y_3\}$ with: $y_0 \rightarrow_{\pi_2} y_1 \rightarrow_{\pi_2} y_2 \rightarrow_{\pi_1} y_3 \rightarrow_{\pi_3} y_4 = y_0$ (with $\pi_2 = (1)$, $\pi_1 = (2 \cdot 2 \cdot 3)$, $\pi_3 = (1 \cdot 4)$). This limit cycle is depicted on the left of Figure 8.22, and its Δ-unfolding (corresponding to 7 points generated by the composing modes of $\pi_2\pi_2\pi_1\pi_3$) on the right.

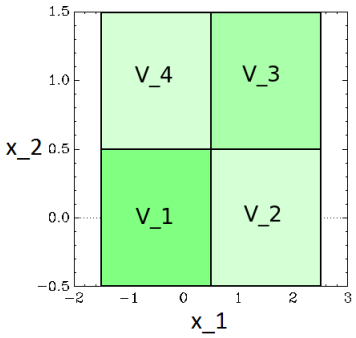


Figure 8.19: Decomposition for the two-tank problem

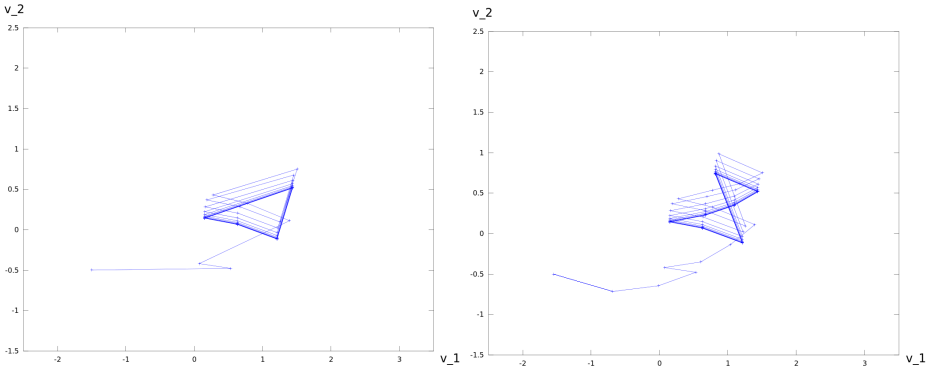


Figure 8.20: Δ-trajectory starting from the bottom left corner of R (left), and its Δ-unfolding (right)

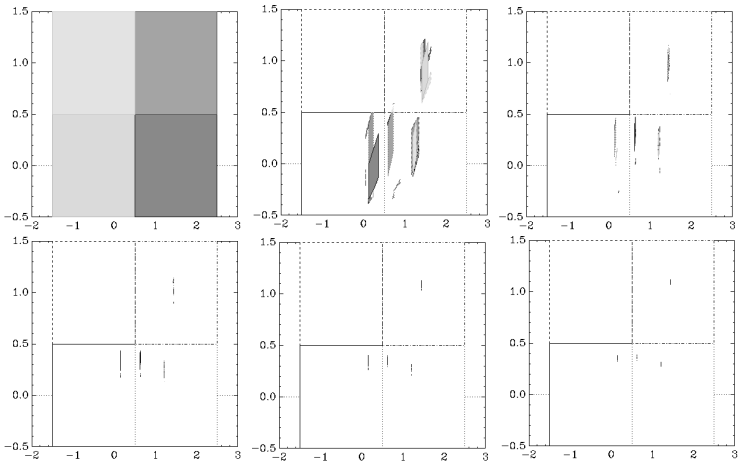


Figure 8.21: Visualization of $Post_{\Delta}^{k}$ for $k = 0, 5, 10, 15, 20, 25$

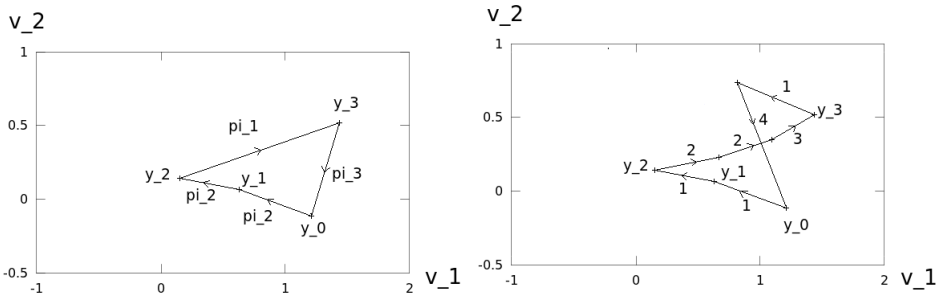


Figure 8.22: Limit cycle for the two-tank example (left), and its Δ-unfolding (right)

8.5 Implementation

The implementation of the method is made of two basic procedures: a procedure Decomposition and described in Section 8.3.1, written in Octave [Oct13], which outputs Δ, and a procedure, called Iteration which constructs R^i_Δ for $i \geq 0$, written in Ocaml [Oca13]. The procedure Decomposition is implemented using zonotopes.

One cannot implement the procedure Iteration using zonotopes because it involves the intersection operator which does not preserve the structure of zonotopes. It is thus implemented using the more general structure of polyhedra using the PPL library [PPL13]. The Iteration procedure receives Δ from module Decomposition and outputs the successive iterations of $Post_\Delta$. The sequence of post sets can also be visualized as an animation (see Figure 8.17).

The examples of decomposition given in this thesis have been performed using the tool MINIMATOR [min]. The multilevel simulations have been performed using a code written in PLECS [PLE13], which is better-suited for these examples, because in PLECS one can enter the electrical circuits under a schematic form and obtain automatically the associated systems of differential equations. The examples have run on a machine equipped with an Intel Core2 CPU X6800 at 2.93GHz and with 2GiB of Ram memory. Some figures of the experiments are listed in Table 8.1.

The first column indicates the name of the example together with its reference in the thesis. The second column indicates the running time to obtain a decomposition, and the third one the numbers of patterns generated to obtain this decomposition[8]. The subsequent columns labeled by N, k, d and n indicate the number of modes, the input parameter of maximal pattern length,

[8]This figure is not available for the multilevel converters because they have been implemented using PLECS, rather than our standard code in Octave (see Chapter 8).

Example	Running time	# patterns	N	k	d	n	contrac.	cycle
Boost (ex. 20)	150 seconds	12113	2	10	4	2	loc.	yes
Helicopter (ex. 21)	≈ 2 hours	≈ 1.5 million	9	6	4	2	no	yes
Heating (ex. 22)	1 second	134	2	2	4	2	glob.	yes
Two-tank (ex. 25)	4 seconds	1423	4	3	1	2	glob.	yes
5-level (Sec. 9.2.1)	3 minutes	-	16	8	1	3	yes	yes
7-level (Sec. 9.2.2)	35 minutes	-	64	32	1	5	yes	yes
9-level (Sec. 9.2.2)	≈ 5 hours	-	256	128	1	7	yes	yes

Table 8.1: Case studies run on MINIMATOR

the input parameter of decomposition depth and the space dimension respectively. Finally, the column 'contrac.' indicates if the example is locally contractive ('loc.'), globally contractive ('glob') or not contractive, the column 'cycle' if the procedure Iteration generates a limit cycle.

8.6 Extensions: Reachability, Sensitivity, Robustness, Nonlinearity

In this chapter, we introduce several novel ways of exploiting different facets of the procedure of invariant generation by decomposition introduced in Chapter 8.

8.6.1 Reachability Control

The *reachability control* problem consists in steering the system from an initial region, say R, to a target region typically containing a reference point, say O. We have seen in Chapter 8 how the iteration of the decomposition procedure may drive the trajectories to a stability subregion of R. If such a region contains O, then we are done. Actually, if we start from a vast region of attraction R, it is possible to interleave the process of generating the successor sets with a process of updating the decompositions We first apply the decomposition procedure to R: if the procedure succeeds, we get a decomposition Δ with $Post_\Delta(R) \subset R$. If $Post_\Delta(R)$ still contains the reference point 0, one re-applies the decomposition procedure to the bounding box $R' = \Box(Post_\Delta(R))$, of $Post_\Delta(R)$. If the procedure succeeds, yielding a new decomposition Δ', one can compute the successor set $Post_{\Delta'}(R') \subset R'$. Now, if $Post_{\Delta'}(R')$ still contains O, one can reiterate the process to $R'' = \Box(Post_{\Delta'}(R'))$, and so on iteratively. One thus produces nested boxes $R, R', R'', \ldots$ and associated decompositions $\Delta, \Delta', \Delta'', \ldots$. The procedure terminates either when the sequence stabilizes ($R^{i+1} \approx R^i$), or when the decomposi-

tion procedure fails, or when R^{i+1} does not contain O. The control induced by the decompositions $\Delta, \Delta', \ldots$ can be used to construct a *reachability controller* that steers the system from R to a region R^i containing O. This is illustrated in Example 26.

Example 26. *We first illustrate the process of iterative decomposition on the boost converter example (see Example 12). We start from a "large" region* $R = [0,10] \times [0,4]$. *We take* $O = (1.8, 1.2)$ *as a reference point. The iteration finds 8 nested decompositions (see Figure 8.23 generated by a script of ours that recursively calls MINIMATOR). A trajectory controlled by such nested decompositions, is given in Figure 8.24 generated by a script of ours. The process is also illustrated on the helicopter motion example (see Example 14), starting from* $R = [-10,10] \times [-10,10]$, *with the origin* $(0,0)$ *as a reference point. The iteration finds 9 nested decompositions (see Figure 8.25 generated by a script of ours that recursively calls MINIMATOR). A controlled trajectory is given in Figure 8.26 generated by a script of ours.*

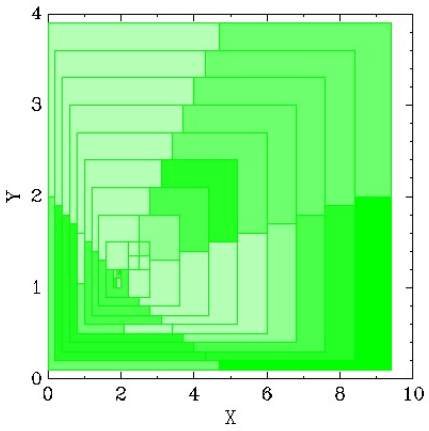


Figure 8.23: Nested decompositions for boost DC-DC converter found by starting from $R = [0,10] \times [0,4]$

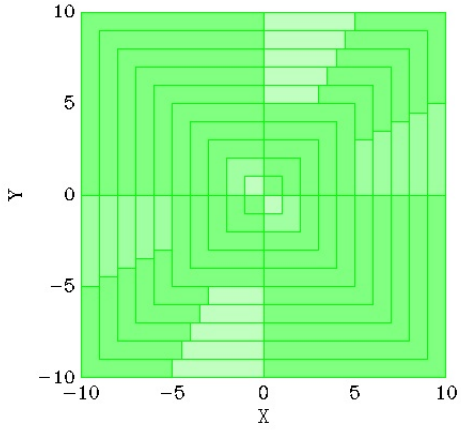


Figure 8.24: Nested decompositions for helicopter motion found by starting from $R = [-10,10] \times [-10,10]$

Actually, since the procedure constructs the successive decompositions in a "blind" manner, without *a priori* consideration for the position of O, it is unable to drive the system to a *close* neighborhood of O. This shortcoming seems

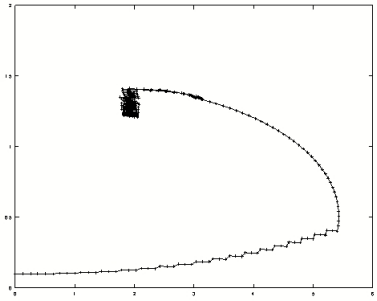


Figure 8.25: Controlled trajectory of boost DC-DC converter starting from $(0, 0.01)$

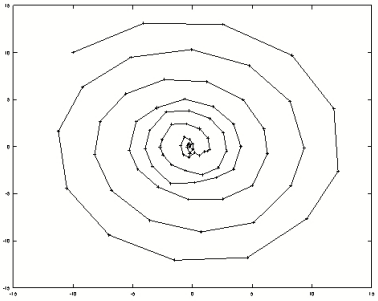


Figure 8.26: Controlled trajectory of helicopter starting from $(-10, 0)$

unavoidable when using a *forward* approach. More precise reachability controllers should implement a *backward* approach, starting from O and using dynamic programming techniques as done, e.g., in [Ber00, LTS99], but this may require an expensive gridding of the state space.

8.6.2 Sensitivity

We have shown in Chapter 8 that, under the control induced by decomposition of a given region R, trajectories trajectories converge to stable *limit cycles* inside R. In the following example, we point out here the sensitivity of limit cycles to parameter variations by showing the evolution of limit cycles in presence of small perturbations of system parameters. As indicated, e.g., in [His01], this suggests that limit cycles are are good candidates for reliable estimation of physical parameters of the system, using an appropriate *inverse approach* (see [Tar05]).

Example 27. *We take the boost DC-DC example with the same region* $R = [1.55, 2.15] \times [1.0, 1.4]$ *as considered in Example 17. The application of the algorithm 7 with* $k = 5$ *and* $d = 1$*, yields a decomposition* Δ *of* R *(see Figure 8.1 of Chapter 8). As depicted in Figure 8.27 generated by our script that simulates runs, the trajectories starting from the four corners of* R*, under the control strategy induced by* Δ *converge to the same limit cycle. A remarkable feature is that, even in presence of (small)* variations *of parameters of the system, the same decomposition* Δ *ensures the* k*-invariance of* R*. In our example, the decomposition* Δ*, originally found for* $r_0 = 1$*, is still* k*-invariant when* r_0 *varies from* 0.965 *to* 1.005*. It follows that the state-dependent control found for* $r_0 = 1$ *still ensures the convergence to limit cycles in* R*, for slight variations of* r_0*. Nevertheless, as shown*

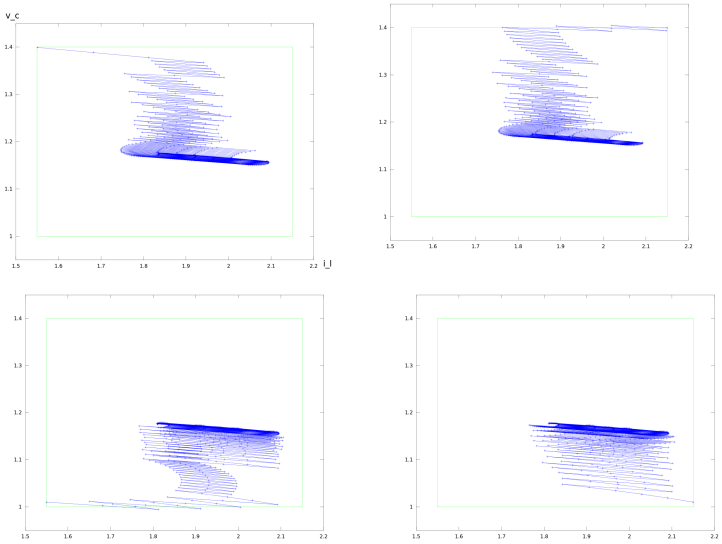


Figure 8.27: Runs starting from the four corners of R, following the control strategy induced by the decomposition, and converging to the same limit cycle

in Figure 8.28 generated by a script of ours, the form, length and position of the limit cycles are very sensitive to the actual value of r_0*: for* $r_0 = 0.965$*, the limit cycle corresponds to the pattern* $(\pi_3\pi_1^3\pi_3\pi_1^2\pi_3\pi_1^3\pi_3\pi_1^3\pi_3\pi_1^3)$ *(with* $\pi_1 = (1\cdot1\cdot2\cdot2\cdot2)$*,* $\pi_3 = (2\cdot1\cdot2)$*); for* $r_0 = 0.975$*, the pattern is* $(\pi_3\pi_1^5)$*, while, for* $r_0 = 1$ *and* $r_0 = 1.005$*, it is just* (π_1)*.*

8.6.3 Robust Safety Control

As explained in Section 7.4.3, zonotopes allow to extend the computation of successor sets in order to account for small perturbations of the system dynamics. The dynamics of the system are now of the form $\dot{x} = A_u x + b_u + \varepsilon$ where ε represents a disturbance vector belonging to a given box Λ centered at the origin, and relations $Post_u$ and $Post_\pi$ are extended, as explained in Section 7.4.3. The decomposition procedure and its enhancement for safety (Section 8.3.3) are then simply adapted by replacing the inclusion test $Post_\pi(W) \subset R$ by $Post_\pi(W, \Lambda) \subset R$. We now give two examples of application of the decomposition procedure (enhanced for safety) in presence of disturbance.

Example 28. *(***Boost converter***). We consider the dynamics of the boost DC-DC converter (see examples 17 and 20) in presence of disturbances* ε *belonging to* $\Lambda = [0,0] \times [\frac{-0.064}{x_l}, \frac{0.064}{x_l}]$*. These disturbances correspond to noise on the input voltage, and represent up to* 8% *of the value of the input voltage. With such dis-*

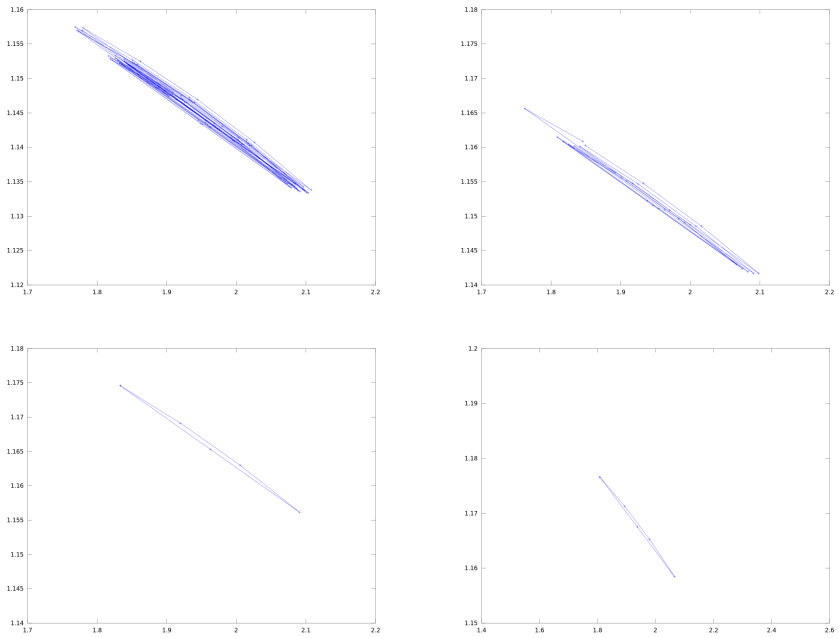


Figure 8.28: Limit cycles for $r_0 = 0.965$ (pattern $(\pi_3\pi_1^3\pi_3\pi_1^2\pi_3\pi_1^3\pi_3\pi_1^3\pi_3\pi_1^3)$) on the upper left, for $r_0 = 0.975$ (pattern $(\pi_3\pi_1^5)$) on the upper right, for $r_0 = 1$ (pattern π_1) on the lower left, and for $r_0 = 1.005$ (pattern π_1) on the lower right

turbances, we are not *able to find a control preserving the safety zone of Example 20 (viz.,* $[1.7,2.0] \times [1.10,1.30]$*). We thus consider a larger (i.e., more tolerant) safety zone defined by* $S' = [1.65,2.05] \times [1.10,1.30]$*. The extended decomposition procedure then succeeds for* $k = 13$ *and a* $d = 5$*: it generates a* k*-invariant decomposition* Δ' *of* R *satisfying* $Unf_{\Delta'}(R) \subset S'$*. The decomposition* Δ' *is depicted in Figure 8.29 generated by MINIMATOR.*[9] *A trajectory starting at* $(1.75,1.26)$*, submitted to perturbation, is depicted in Figure 8.30 generated by our script that simulates runs.*

Example 29. *(***Helicopter Motion***) As done in [DLHT11], we will now solve the control problem with bounded disturbances to take into account a potential real-life environment. We consider the same regions* $R = [-0.3,0.3] \times [-0.5,0.5]$ *and* $S = [-0.4,0.4] \times [-0.7,0.7]$ *as in Example 21, and add the disturbance* $\varepsilon \in \Lambda = [-0.02,0.02] \times [-0.1,0.1]$*. The extended decomposition procedure succeeds, and generates a* k*-invariant decomposition* Δ' *with* $Unf_{\Delta'}(R) \subset S$*. The de-*

[9] The corresponding patterns are: $\pi_1 = (1122121222)$, $\pi_2 = (1122122121222)$, $\pi_3 = (21121222)$, $\pi_4 = (12121222)$, $\pi_5 = (122)$, $\pi_6 = (2)$, $\pi_7 = (12)$, $\pi_8 = (12)$, $\pi_9 = (12)$, $\pi_{10} = (12)$, $\pi_{11} = (1)$, $\pi_{12} = (1)$, $\pi_{13} = (1)$, $\pi_{14} = (12)$, $\pi_{15} = (12)$, $\pi_{16} = (21221)$.

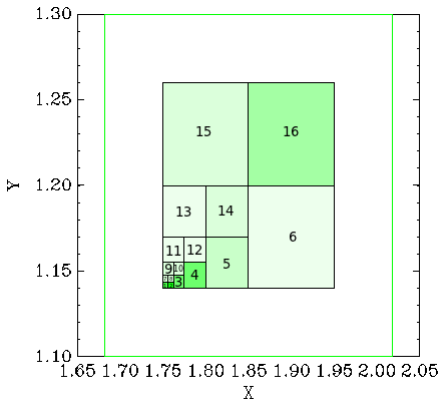


Figure 8.29: k-invariant decomposition for boost converter with disturbances

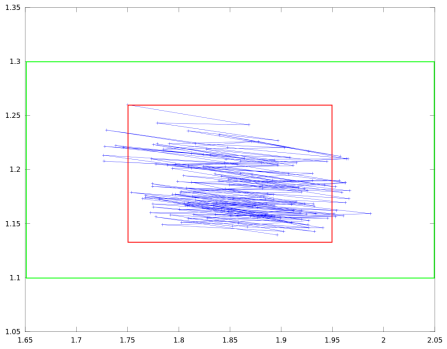


Figure 8.30: Unfolded Δ-trajectory of the boost converter with disturbances in plane (i_l, v_c), starting at $(1.75, 1.26)$ (inner box: $R = [1.75, 1.95] \times [1.14, 1.26]$, outer box: $S' = [1.65, 2.05] \times [1.1, 1.3]$)

composition Δ' is depicted in Figure 8.31 generated by MINIMATOR. A trajectory starting at point $(-0.3, 0.5)$, submitted to disturbance, is depicted in Figure 8.32 generated by a script of ours.

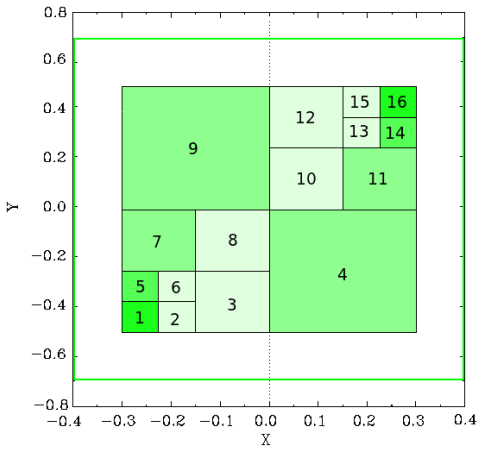


Figure 8.31: k-invariant decomposition for helicopter motion with disturbance

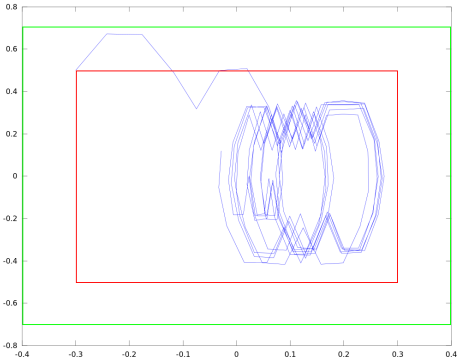


Figure 8.32: Unfolded Δ-trajectory of helicopter motion with disturbance in plane $(x, \dot{x})$, starting at $(-0.3, 0.5)$ (inner box: R, outer box: S)

8.6.4 Nonlinearity

The basic decomposition procedure, explained in Section 8.3.1, is quite general, and does not suppose that the successor operator $Post_\pi$ is linear or affine. However, in the case where $Post_\pi$ is an affine function, the computation can be done in an *exact* manner. We now sketch out how to modify the decomposition procedure in case of non-affine dynamics. The process is inspired by what has been done for handling disturbance (Section 8.6.3). Following [ASB08], we compute (an overapproximation of) the successor sets using *local linearizations* of the system, and enlargement of the linear images by addition of *error* intervals. We will consider here a system governed by a *unique* equation of the form $x(t+\tau) = f(x(t))$ where f is a polynomial. The set U is thus reduced to a single element ($U = \{1\}$). A pattern π associated to a subregion V, is now just an *integer* indicating the number of times f should be applied when the state is in V. We put $f(x)$ under the form $f(x) = f_{lin}(x) + Q(x)$, where $f_{lin}(x)$ is a polynomial of order 1, and Q a sum of monomials of order greater than or equal to 2. Along the lines of Section 7.4.3, we can compute a local over-approximation of f by considering the polynomial subpart $Q(x)$ as a perturbation. We have:

Lemma 14. *Consider a function f defined by: $f(x) = f_{lin}(x)+Q(x)$, where $f_{lin}(x)$ is a 1st-order polynomial of the form $d + Cx$, and $Q(x)$ a sum of polynomials of order greater than equal to 2. Given a zonotope $Z :< c, G >$, we have:*

$$f(Z) \subset f_{lin}(Z) + Z_\Lambda$$

with:

- $f_{lin}(Z) =< f(c), CG >$
- $Z_\Lambda =< 0, \begin{pmatrix} \varepsilon_1(Z) & 0 & \dots & 0 \\ 0 & \varepsilon_2(Z) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \varepsilon_n(Z) \end{pmatrix} >$
  with ($1 \le i \le n$): $\varepsilon_i(Z) = \max_{x \in Z}(|Q_i(x) - Q_i(c)|)$.

Using a repeated application of the lemma, one can compute an over-approximation of $f^\pi(Z)$ of the form $f^\pi_{lin}(Z) + Z^\pi_\Lambda$. This lemma allows to modify the Decomposition procedure as follows: the inclusion test $Post_\pi(W) \subset R$, which corresponds here to a test of the form $f^\pi(W) \subset R$, is replaced by test $f^\pi_{lin}(W) + W^\pi_\Lambda \subset R$. What remains to do is to find an upper bound of $\varepsilon_i(W)$, that is an upper bound of $(|Q_i(x) - Q_i(c)|)$ over W, for each $1 \le i \le n$. We now explain on two standard examples (taken from [ABSTDG12]) how to compute such upper bounds. The decomposition procedure is then iterated in order to construct an attractor, which is an over-approximation of R^*_Δ.

Example 30. *(**Van der Pol oscillator**). The dynamics of the Van der Pol oscillator are the following:*

$$x(t+\tau) = \begin{pmatrix} 1 & \tau \\ -\tau & 1+\tau \end{pmatrix} x(t) + \begin{pmatrix} 0 \\ x_1(t)^2 x_2(t)\tau \end{pmatrix}.$$

When linearized to a point $c \in \mathbb{R}^2$, this gives:

$$x(\tau) = \begin{pmatrix} 1 & \tau \\ -\tau & 1+\tau \end{pmatrix} x(t) + \begin{pmatrix} 0 \\ -c_1^2 c_2 \tau \end{pmatrix}.$$

Thus, we have $f_{lin}(Z) = \begin{pmatrix} 1 & \tau \\ -\tau & 1+\tau \end{pmatrix} Z + \begin{pmatrix} 0 \\ -c_1^2 c_2 \tau \end{pmatrix} = \begin{pmatrix} 1 & \tau \\ -\tau & 1+\tau(1-c_1^2) \end{pmatrix} x(t).$

It is easy to see that for a box $V \subset \mathbb{R}^2$ we are making an error of at most 0 on the x axis and $|(c_1^2 - (c_1 + G_{1,2} + G_{2,2})^2|\tau$ on the y axis. Thus we need to enlarge any image of a zonotope $Z =< c, G >$ by 0 on the x-axis and $\tau|(C_1^2 - (C_1 + G_{1,2} + G_{2,2})^2|$ on the y-axis, i.e.:

$Z_\Lambda =< 0, \begin{pmatrix} 0 & 0 \\ 0 & |(C_1^2 - (C_1 + G_{1,2} + G_{2,2})^2|\tau \end{pmatrix} >.$

The Decomposition procedure is applied to $R = [-3,3] \times [-3,3]$ and $\tau = 0.01$ (with parameters $k = 30$, $d = 7$). At boxes located around the center of R, the length of patterns is 1 while in the lower left and upper right edges, the length is up to 30. The result of the Decomposition is depicted in the left part of Figure 8.33 generated by MINIMATOR and (an overapproximation of) the attractor set R_Δ^* in the right part.

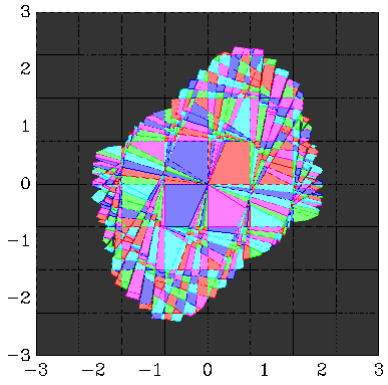


Figure 8.33: Decomposition for the Van der Pol oscillator (left) ; R_Δ^j for $j = 30$ (right)

Example 31. *(**FitzHugh-Nagumo Neuron**). The dynamics of the FitzHugh-Nagumo neuron are the following:*

$$x(\tau) = \begin{pmatrix} 1+\tau & -\tau \\ 0.08\tau & -0.0064\tau+1 \end{pmatrix} x(t) + \begin{pmatrix} -x_1(t)^3\tau/3 + 0.875\tau \\ 0.056\tau \end{pmatrix}$$

When linearized to a point $c \in \mathbb{R}^2$, this gives:

$$x(\tau) = \begin{pmatrix} 1+\tau & -\tau \\ 0.08\tau & -0.0064\tau+1 \end{pmatrix} x(t) + \begin{pmatrix} -c_1^3\tau/3 + 0.875\tau \\ 0.056\tau \end{pmatrix}$$

It is easy to see that for a box $V \subset \mathbb{R}^2$ we are making an error of at most $max_{x\in V}(\frac{|x_1^3-c_1^3|}{3})\tau$ on the x axis and 0 on the y axis. Thus we need to enlarge any image of a zonotope Z by $max_{x\in Z}(\frac{|x_1^3-c_1^3|}{3})\tau$ on the x-axis and 0 on the y-axis, i.e.:

$$Z_\Lambda = < 0, \begin{pmatrix} max_{x\in Z}(\frac{|x_1^3-c_1^3|}{3})\tau & 0 \\ 0 & 0 \end{pmatrix} >.$$

The Decomposition procedure is applied to $R = [-2.5, 2.5] \times [-0.5, 2.5]$ and $\tau = 0.1$ (with parameters $k = 30$, $d = 7$). For boxes located around the center of R, the length of patterns is 1 while in the lower left and upper right corners, the length is up to 22. The result of the Decomposition is depicted in the left part of Figure 8.34 generated by MINIMATOR and (an overapproximation of) the attractor set R_Δ^* in the right part.

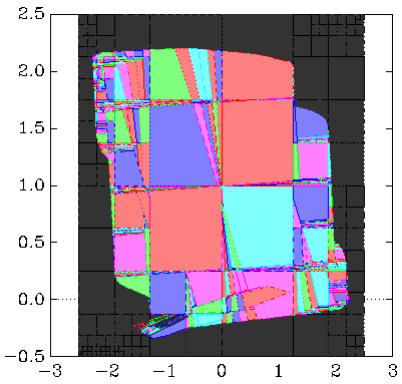


Figure 8.34: Decomposition for the FitzHugh-Nagumo Neuron (left) ; R_Δ^j for $j = 30$ (right)

We have thus sketched out how to construct attractors of polynomial dynamical systems by over-approximating the sub-polynomial subpart of order greater than 1. So far, the over-approximation is done in an *ad hoc* fashion for each specific example. It would be interesting to use a general method of over-approximation based, e.g., on the notion of Lagrange remainder (see [ASB08]).

8.7 Discussion

The class of $\mathscr{S}^2$-systems is a subclass of *affine hybrid systems* [Hen96] where the discrete transitions only happen at instants that are integer multiple of τ. Alternatively, one can consider $\mathscr{S}^2$-systems as models capturing only continuous transitions of duration τ.

Using zonotopes for enclosing the uncertainty, one can avoid the *wrapping effect*, which leads to exponential fast growing enclosures when using iteratively naive approximation (like bounding box).

The method of proving controlled invariance presented here, based on a decomposition of the state space, is original. Moreover, as previously stated, our method uses only forward computation which is of prime interest when considering contractive systems. The multilevel converter case study could not have been achieved using backward methods.

As pointed out above, the limit cycle generated is not necessarily a minimal invariant. Actually, it is difficult to define an appropriate notion of size for a minimal invariant set.

The reachability approach, sketched out in Section 8.6.1, does not take into account the question of time-optimality. The *time-optimal control* problem, consists in steering, in minimal time, the state of the system to a desired target while keeping the system safe along the way.

8.8 Related Work

The semantics of sampled switched systems given here originates from the work of Antoine Girard, Giordano Pola and Paulo Tabuada (see, e.g., [Tab09]). We have simplified here their formalism by removing the notion of *output* and *observation* sets. What has been called here *sampled switched system*, denoted by Σ, corresponds actually in [Tab09, Chapter 11]to the notion of *symbolic system associated with a switched affine system* Σ, denoted by $S_\tau(\Sigma)$.

The idea of approximating the state trajectories of dynamical systems using zonotopes comes from [Küh98]. Zonotopes have recently received many applications in the domain of hybrid systems with uncertainty such as reachability analysis [ASB07].

This method presents some similarities with the method of *box invariance* of [ATS09] which exhibit rectangular invariant subregion of affine hybrid systems containing an equilibrium point. The process of state decomposition by dichotomy (or "bisection") has been used in [JKDW01] for the purpose of *set inversion* and applied to a robust control and stability analysis.

The presence of limit cycles in switched systems has often been observed

in the context of power electronics (see, e.g., [PG09]). In the framework of discrete LTI with quantized input, Picasso and Bicchi have used hypercubes in order to provide a lower bound for the minimal feasible size of an invariant set [PB08]. Various methods are classically used for proving their existence and stability: Lyapunov techniques (see, e.g., [RL00][BRC05]); Poincaré map technique (see, e.g., [Gon03, His01]); sensibility functions [FRL06], or describing functions [San93].

The time-optimal control is classically solved using dynamic programming and a backward procedure (see [Ber00]). A solution for $\mathscr{S}^2$-systems, based on approximate bisimulation, is given in [Gir10].

The idea of exploiting trajectory sensitivities in order to solve inverse problems, sketched out in Section 8.6.2, comes from [His01], where it is explained how to use the measurements of disturbance effects to improve the estimates of parameters of power systems.

Chapter 9

Application to Multilevel Converters

Power converters play an important role in the field of renewable energy: they are used to connect renewable sources to power-grids, optimize the efficiency of solar panels and wind generators. Switched control has gained much attention recently in the field of high order converters, due to its property of being easily implemented. In some topologies, there is however a dramatic increase of the number of switches, which entails an increasing number of degrees of freedom, and complicates the controller design (see [CPPMT09]). There is therefore a niche of application for formal methods in order to produce correct-by-design control methods.

In this chapter, we consider the design of control policies for power converters with 5 and 7 levels. The objective is to design a switching signal forcing the output voltage to be a "quantized" sinusoidal signal while ensuring that the voltages across the capacitors in the power converter remain within predefined safety ranges. The staircase waveform is achieved by controlling 4 (resp. 6) switches, which are used to divide the incoming voltage into to paths and produce different levels of incoming voltage with the help of 3 (resp. 5 capacitors). We adapt the decomposition procedure explained in Chapter 8, in order to synthesize a controller that guarantees that the electrical state parameters will always stay within a predefined safe zone of variations. The synthesized controllers have been validated on real hardware on prototypes built by SATIE Electronics Laboratory.

Contributions Our contributions in this chapter are: the study of multilevel converters using the Decomposition Procedure in Section 9.2, and the implementation of the controllers on the prototype for the physical experimentations of Section 9.3.

Outline of the chapter In Section 9.1, we explain the principle and architecture of multilevel converters. In Section 9.2, we apply the decomposition method to the control of 5-level and 7-level converters, and give numerical simulations. In Section 9.3, we present physical experimentations done with a prototype built by SATIE Laboratory.

9.1 Multilevel Converters

The general function of a multilevel power converter is to synthesize a desired voltage from several levels of DC voltage. For this reason, multilevel power converters can easily provide the high power required by large electric drive systems. Schematically, a multilevel converter is made of capacitors and switching cells (as well as opposite switching cells which are in complementary positions). According to the positions of the cells (the high-side switch conducting position is indicated by 1 and the lowside switch conducting position by 0), one is able to fraction the load voltage. By controlling the global position position of the switches during a simple fixed time-stepping procedure, it is then possible to generate a staircase voltage with levels that approximates a triangular or a sinusoidal waveform (see Figure 9.1 for $\ell = 5$). The problem which arises is to select the appropriate switching control strategy among a number of combinations of switch positions which increases exponentially with the number of levels (and pairs of switches). A crucial additional difficulty comes from the fact that, in order to be admissible, the control of the switching cells must guarantee that the voltages across the cell-capacitors are constrained within a certain range defined by the device blocking voltage rating. The control must then guarantee a *safety property*, called "capacitor voltage balancing": the voltage of each individual capacitor should stay inside a limited predefined interval.

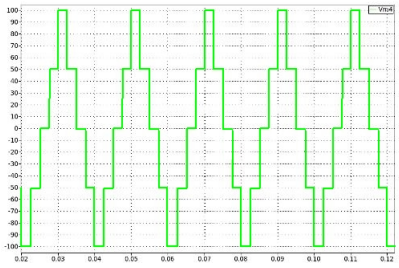


Figure 9.1: Staircase output voltage waveform for a 5-level converter

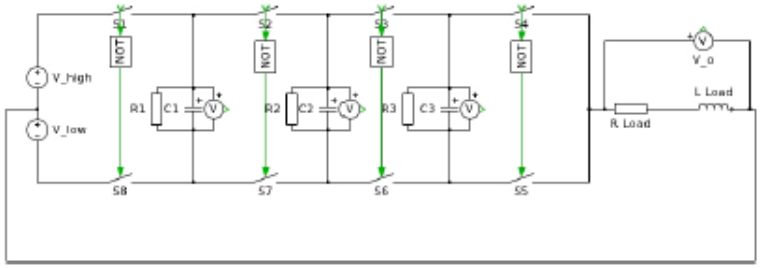


Figure 9.2: Electrical scheme of a 5-level converter

9.2 Application of the Decomposition Procedure

We apply a simplified version of the decomposition procedure given in Section 8.3.1 to the context of this case study. The procedure can be simplified here because, as explained below, only a restricted number of patterns allows to produce a staircase output signal of ℓ levels. These admissible patterns correspond to paths in a pre-defined graph. They have all the same length $k = 2(\ell - 1)$. Besides, the procedure succeeds by simple bisection of the input control box R into $2^{\ell-1}$ sub-boxes (which corresponds to value $D = 2$ for input depth parameter of Algorithm 7).

9.2.1 5-level Converter

There are different possible topologies for multilevel power converters: neutral-point clamped, cascaded H-bridge, flying capacitor... We focus here on the flying capacitor topology [MF92]. The electrical scheme of a 5-level converter is given in Figure 9.2. There are 4 pairs of switching cells S_1, S_2, S_3, S_4 and 3 capacitors $C_1, \ldots, C_3$. The state of the system is $x(t) = [v_1(t)\ v_2(t)\ v_3(t)\ i(t)]^T$ where $v_i(t)$ is the voltage across C_i $(1 \leq i \leq 3)$ and $i(t)$ is the current flowing in the circuit. The duration of a cycle is $T = 8\tau$. The *mode* of the system is characterized by the value (0 or 1) of the switching cells, i.e., by the value of vector $S = [S_1\ S_2\ S_3\ S_4]^T$.[1] There are thus $2^4 = 16$ modes. A mode S induces an output voltage of value $-v_{low} + (\Sigma_{i=1}^{4} S_i) v_{high}/2$, where v_{low} and v_{high} are the input voltages of low level and high level respectively. The system thus outputs 5 different levels of voltage which go from $-v_{low}$ up to $+v_{high}$ with steps at $-1, -\frac{1}{2}, 0, \frac{1}{2}, 1$. The ideal value v_i^* of the voltage across capacitor C_i $(1 \leq i \leq 3)$ depends on the values of v_{low} and v_{high}. Here we use: $v_{low} = v_{high} = 100V$,

[1] Besides, we have: $S_5 = \neg S_1$, $S_6 = \neg S_2$, $S_7 = \neg S_3$ and $S_8 = \neg S_4$.

and $v_1^* = 150V$, $v_2^* = 100V$, $v_3^* = 50V$. The 5-level converter can be seen as a switched system. Given a mode S, the associated dynamics is of the form $\dot{x}(t) = A_S x(t) + b_S$ with:

$$A_S = \begin{pmatrix} -\frac{1}{R_1 C_1} & 0 & 0 & \frac{S_1 - S_2}{C_1} \\ 0 & -\frac{1}{R_2 C_2} & 0 & \frac{S_2 - S_3}{C_2} \\ 0 & 0 & -\frac{1}{R_3 C_3} & \frac{S_3 - S_4}{C_3} \\ \frac{S_2 - S_1}{L_{Load}} & \frac{S_3 - S_2}{L_{Load}} & \frac{S_4 - S_3}{L_{Load}} & -\frac{R_{Load}}{L_{Load}} \end{pmatrix} \text{ and } b_S = \begin{pmatrix} 0 \\ 0 \\ 0 \\ S_1 \frac{v_{high} + v_{low}}{L_{Load}} \end{pmatrix}$$

By controlling the modes at each sampling time, one can synthesize a 5-level staircase function. Not all the transitions between modes are admissible: we allow to switch only one (pair of) cell(s) at a time. The graph of admissible transitions during a cycle is depicted in Figure 9.3. The nodes of the graph are labeled by the modes. Each path represents a possible sequence of control for 1 cycle, leading from voltage $-v_{low}$ (state 0000) to voltage $+v_{high}$ (state 1111) through voltages $-\frac{1}{2}.v_{low}$, 0, $\frac{1}{2}.v_{high}$ then back to voltage $-v_{low}$ (state 0000) through voltages $\frac{1}{2}.v_{high}$, 0, $\frac{1}{2}.v_{low}$. There are thus 576 possible sequences of control for generating a 5-level staircase signal on 1 cycle. These sequences of control correspond to patterns of length 8, denoted by $\pi_1, \ldots, \pi_{576}$. The control problem is now to find a strategy for deciding, at each beginning of cycle, which π_i ($1 \leq i \leq 576$) to apply in order to maintain all the capacitor voltages within a predefined limited zone.

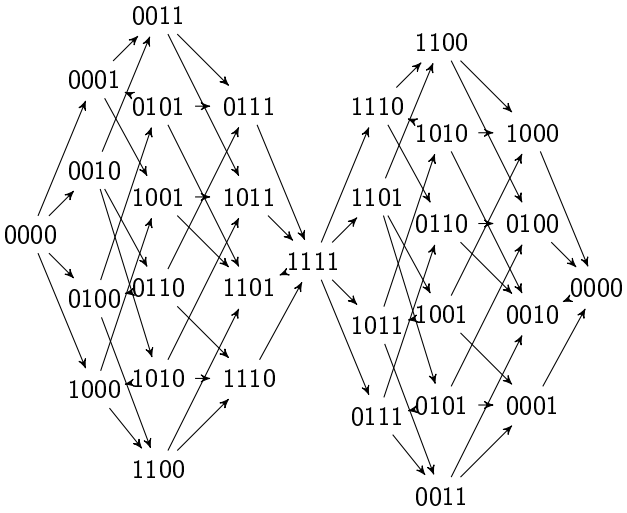


Figure 9.3: Transition graph corresponding to a cycle of 5-level staircase signal

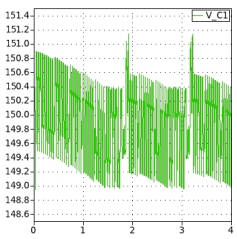
We will use the numerical values: $R_{Load} = 50\Omega$, $C_1 = C_2 = C_3 = 0.0012$F, $L_{Load} = 0.2$H, $R_1 = R_2 = R_3 = 20{,}000\Omega$, $T = 8\tau = 0.02s$ (which corresponds to

a frequency of 50Hz). The 5-level inverter outputs ideally a staircase waveform with an amplitude of 200V, centered around 0V. We consider that a variation of $\pm 5V$ is admissible as it represents a variation of 10% on the least charged capacitor C_3. It is interesting to notice that at each beginning of a cycle the value of i is null. This suggest to look for a state-dependent control which depends only on the capacitor voltages v_1, v_2, v_2, and not on the value of i. We will thus focus on the voltage dimensions of the control box R and disregard its intensity dimension. For R, we take $R = [145,155] \times [95,105] \times [45,55]$, which corresponds to a product of intervals centered around the ideal values with a variation of $\pm 5V$ (i.e., 10% of the least charged capacitor C_3). For S, we take $R + \varepsilon$ with $\varepsilon = 1V$, which means that we have an additional tolerance of $\pm 1V$ for the fluctuations occurring between two beginnings of cycle. The decomposition procedure is thus adapted as follows. We decompose R into 8 subsets V_i of equal size:

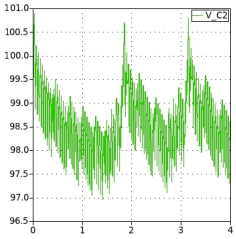
- $V_1 = [145,150] \times [95,100] \times [45,50]$
- $V_2 = [145,150] \times [95,100] \times [50,55]$
- $V_3 = [145,150] \times [100,105] \times [45,50]$
- $V_4 = [145,150] \times [100,105] \times [50,55]$
- $V_5 = [150,155] \times [95,100] \times [45,50]$
- $V_6 = [150,155] \times [95,100] \times [50,55]$
- $V_7 = [150,155] \times [100,105] \times [45,50]$
- $V_8 = [150,155] \times [100,105] \times [50,55]$

Using a standard random generate-and-test program, we find patterns π_j ($1 \leq j \leq 8$), which, applied to points of V_j, generates points that are all contained in S. The patterns π_js correspond to the following paths of the transition graph:
- π_1: (0000 → 0001 → 0101 → 1101 → 1111 → 1101 → 0101 → 0001 → 0000)
- π_2: (0000 → 0100 → 0101 → 1101 → 1111 → 1101 → 0101 → 0100 → 0000)
- π_3: (0000 → 0001 → 0011 → 1011 → 1111 → 1011 → 0011 → 0001 → 0000)
- π_4: (0000 → 0010 → 0011 → 1011 → 1111 → 1011 → 0011 → 0010 → 0000)
- π_5: (0000 → 1000 → 1010 → 1110 → 1111 → 1110 → 1010 → 1000 → 0000)
- π_6: (0000 → 1000 → 1100 → 1101 → 1111 → 1101 → 1100 → 1000 → 0000)
- π_7: (0000 → 0100 → 0110 → 0111 → 1111 → 0111 → 0110 → 0100 → 0000)
- π_8: (0000 → 1000 → 1010 → 1011 → 1111 → 1011 → 1010 → 1000 → 0000)

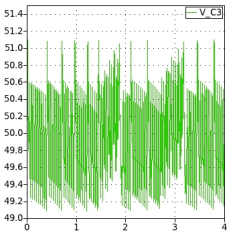
We present in Figures 9.4 and 9.5 a numerical simulation of this controller on the system starting from the point $v_1(0) = 150V$, $v_2(0) = 100V$, $v_3(0) = 50V$ and $i(0) = -3A$. This simulation has been performed using tool PLECS [PLE13]. One can see on the simulation that the system state always stays inside S.



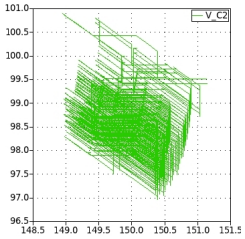
(a) Voltage $v_1 = f(t)$



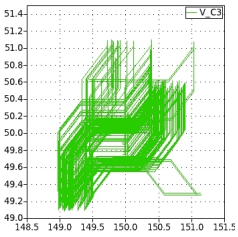
(b) Voltage $v_2 = f(t)$



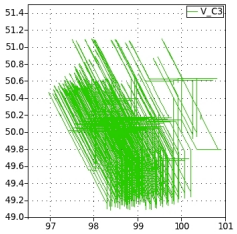
(c) Voltage $v_3 = f(t)$



(d) Plane $v_2 = f(v_1)$



(e) Plane $v_3 = f(v_1)$

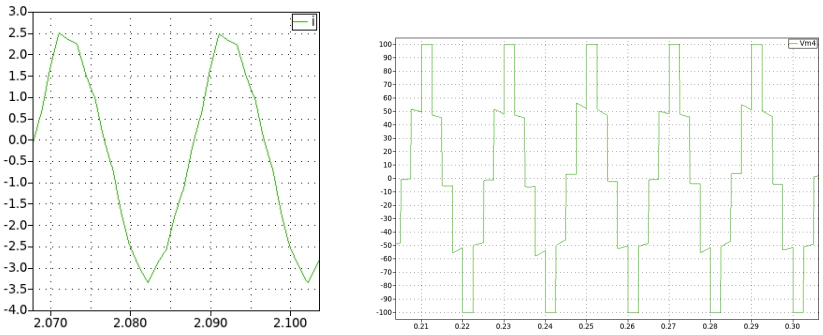


(f) Plane $v_3 = f(v_2)$

Figure 9.4: Capacitor voltages

9.2.2 7-level Converter

The flying capacitor architecture is generic. We now consider the case of a ℓ-level converter with $\ell = 7$. There are now 6 pairs of switching cells and 5 capacitors $C_1, \ldots, C_5$. The state of the system is $x(t) = [v_1(t)\ v_2(t)\ v_3(t)\ v_4(t)\ v_5(t)\ i(t)]^T$ where $v_i(t)$ is the voltage across C_i $(1 \leq i \leq 5)$ and $i(t)$ is the current flowing



(a) Current i (b) Output voltage v_o

Figure 9.5: Current and output voltage

in the circuit. The generated waveform now goes from $-v_{low}$ up to $+v_{high}$ with steps at $-v_{low} + i v_{high}/3$ for $i = 0,\ldots,6$, and the duration of a cycle is $T = 12\tau$. There are now 518,400 possible sequences of control (patterns) for generating an ℓ-level staircase signal on 1 cycle. We used the following values for the system constants: output at 50Hz,[2] capacitances of $0.1F$, resistor values 50Ω, inductor values $0.137H$, $v_{low} = v_{high} = 300V$. Ideally, the output is thus a staircase waveform with an amplitude of 600V, centered around 0V, and the ideal values v_i^* of the capacitor voltages of the capacitor C_i are given by: $v_1^* = 500V$, $v_2^* = 400V$, $v_3^* = 300V$, $v_4^* = 200V$, $v_5^* = 100V$. For R, we take $R = [495,505] \times [395,405] \times [295,305] \times [195,205] \times [95,105]$, which corresponds to a product of intervals centered around the ideal values with a variation of $\pm 5V$ (i.e., 5% of the least charged capacitor C_5). For S, we take $R + \varepsilon$ with $\varepsilon = 1V$, which means that we have an additional tolerance of $\pm 1V$ for the fluctuations occurring between two beginnings of cycle. The decomposition of R into sub-zones $\{V_i\}_{i\in I}$ and the corresponding set of patterns $\{\pi_i\}_{i\in I}$ are given in [FFL+12]. We present in Figures 9.6 and 9.7 a numerical simulation of the controlled system starting from the point $v_1(0) = 500V$, $v_2(0) = 400V$, $v_3(0) = 300V$, $v_4(0) = 200V$, $v_5(0) = 100V$ and $i(0) = -2.5A$. One can check on the simulation that the system state always stays inside S.

We have also performed experiments with a ℓ-level converter for $\ell = 9$. We have been able to obtain a decomposition after 5 hours of running time, but we are clearly at the limit of the existing implementation.

[2] which corresponds to $T = 12\tau = 0.02s$

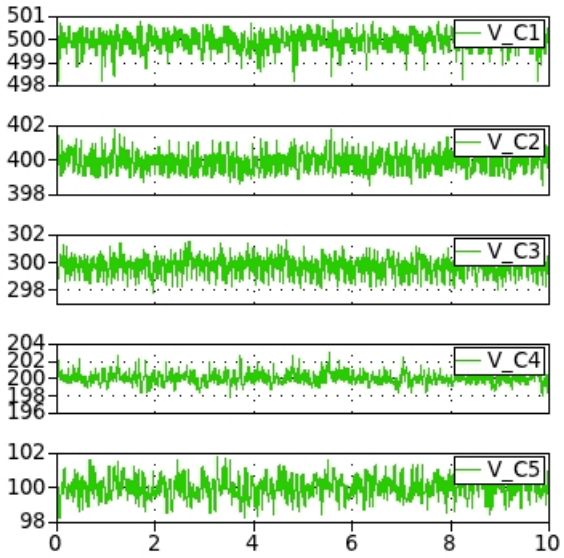
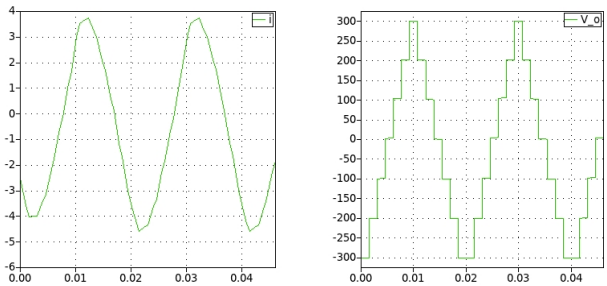


Figure 9.6: Capacitor voltages



(a) Current i (b) Current v_o

Figure 9.7: Current and output voltage

9.3 Physical Experimentations

A prototype of the 5-level flying capacitor has been realized by the SATIE Laboratory in order to test our control strategy on an actual system. See Figure 9.8 for a picture of the system. Our control strategy was applied to the system via Simulink and a dSpace® interface. The results are presented in Figure 9.9 for the output voltage and the capacitor charges. In Figure 9.10, we present the same results but with a larger scale on the capacitor voltage to see the fluctuations around the reference values. As we can see, the experimental results are very closed to those obtained by simulation with PLECS [PLE13]. In Figure 9.11, we represent the output voltage together with the current (after appropriate resiz-

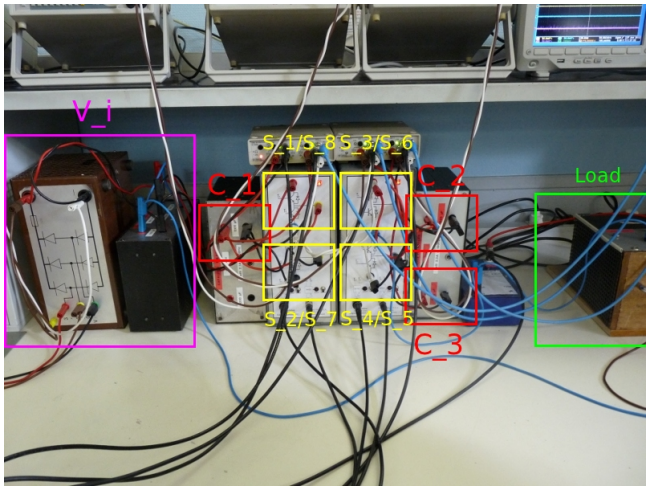


Figure 9.8: Prototype built by SATIE

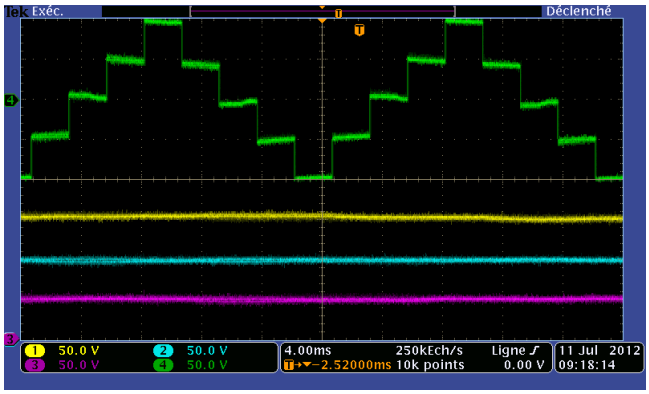


Figure 9.9: Output voltage and capacitor voltages

ing) flowing the load. During the experimentations, we have successfully tested the robustness of the controller in presence of the following perturbations:

1. The ideal voltage source as input is no longer ideal but its values fluctuate around the reference value.

2. The system does not start from the reference valuations for the capacitor voltages and the input voltage, but the input voltage increases gradually until reaching the desired value while the capacitor are initially discharged.

3. We apply the same pattern during two consecutive cycles (instead of updating the pattern at the end of the first cycle).

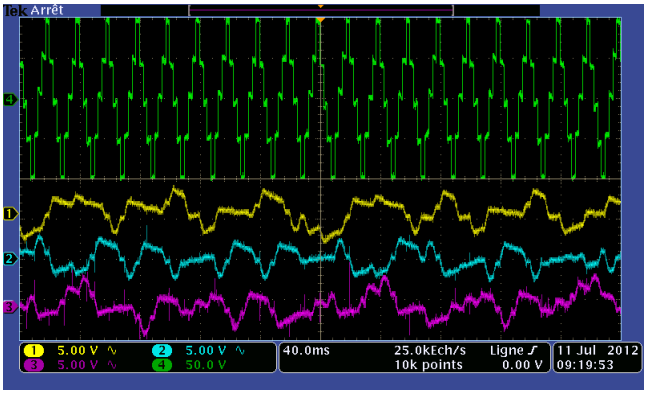


Figure 9.10: Zoom of output voltage (above) and capacitors voltages (below)

Figure 9.11: Output voltage and current (after appropriate resizing) in the circuit

4. We use a time-varying period T of cycle (instead of a constant one), and check the preservation of the capacitor voltages balance. The result of this experiment is depicted in Figure 9.12.

Although these preliminary tests of robustness are promising, they need to be consolidated, in particular in presence of significant variations of resistor loads. Besides, although the principle of the method is general, it also suffers from an exponential increase of complexity when the level ℓ grows: the method reaches the limit for $\ell = 9$, which corresponds to a dimension $n = 7$ of the state space.

Figure 9.12: Output voltage (above) and capacitor voltages (below) in presence of time-varying period T

9.4 Discussion

We have synthesized a state-dependent control that involves only the subset of the state vector related to the voltage information, but ignores the intensity component. This is interesting because for practical applications, a current sensor is not always desired (see [DTOC09]). More generally, control should use, as far as possible, only a subset of the state vector because measuring all signals in high order converters becomes prohibitively expensive.

The method can be easily refined in order to generate sinusoidal-like output signals rather than the triangular-like output signals, as done here: it suffices to adjust the switching instants within the period T of the cycle, instead of using uniformly τ.

Chapter 10

Conclusion and Perspectives

Switched systems are now commonly used in industrial domains such as power electronics or automotive industry. These systems are easily programmable and allow for flexible design of the controlled components. However, with the growing number of switches, the task of control synthesis becomes challenging. Traditional methods based on extensive testing face difficulties to prove correctness of the control designed at hand. This opens the path for formal methods which aimed at the synthesis of *correct-by-design* control software. We have focused in this part of the thesis mainly on two issues of correctness for controllers: *safety* and *stability*. We have seen that the safety problem is closely related to the synthesis of a *maximal controlled invariant* while the stability problem is related to the synthesis of a *minimal controlled invariant*.

The synthesis of a maximal controlled invariant is classically done by a fixed point iteration which is applied in a *backward* manner for computing successively the predecessor reachable sets. The procedure terminates when the state space is finite. Thus, the most common approach to provide correct-by-design synthesis techniques is *indirect*: it first converts the infinite state model into a finite state abstraction, then the controller synthesized at the abstract level is carried through the original infinite state model at the price of a certain approximation. However, in order to keep an acceptable precision, the method imposes a gridding of the state space which entails an explosion of the number of states at the abstract level. An other approach is to work *directly* at the infinite state level. As already mentioned, the backward iteration procedure is thus not guaranteed to terminate, and one has to *under-approximate* the symbolic reachable states generated in order to overcome this problem. However, here again, if one wants to keep an acceptable precision, the number of states generated often becomes intractable, especially in the case of higher dimensional systems.

We have therefore proposed an alternative approach which is *forward ori-*

ented, and works directly on the infinite state level. In order to make this approach tractable, we make some assumptions, looking at the discrete-time version of the dynamics, assuming that the dynamics are affine and a fixed switching frequency. These assumptions are realistic in the context of power electronics. The method interleaves the decomposition of the operating space into subregions with the computation of patterns mapping these subregions inside the operating space. The method can be used for synthesizing safety controllers, as exemplified on the case studies of multilevel converters. Under further assumptions (essentially, local stability), this method is able to stabilize the system around identified limit cycles. We have finally sketched out some possible extensions of the Decomposition procedure in order to address issues such as reachability control, robust control and nonlinear dynamics.

An important topic of current research is to synthesize controllers that satisfy *simultaneously* multiple objectives. When safety is among the objective, a typical method is to construct the *maximally permissive* safety controller, then to refine it in order to complete secondary objectives. As usual, in practice, things are often more difficult, and techniques of synthesis and verification may have to be combined in an *ad hoc* manner. For example, in the oil pump system studied in [CJL+09], the problem is to design an operating cycle that meets not only safety requirements (for maintaining the oil level in a given interval), but also robustness (for taking into account imprecisions on measures of volume or time), and performance (for minimizing the oil accumulated during each cycle). Such a goal is achieved using a clever combination of analysis tools (UPPAAL-TIGA [BCD+07] for synthesis, PHAVER [Fre08] for verification, and SIMULINK [Tea08] for simulation). In the domain of hybrid systems, many tools of control synthesis share indeed many common features with tools of verification: they rely on the same techniques of compact representation of states, and efficient construction of reachable sets. The tool d/dt [ADM02] can thus be used to solve safety verification problems as well as safety switching controller synthesis. Other tools such as SPACEEX [FLGD+11] and IMITATOR [AS13] can be similarly used both for verification and synthesis problems of hybrid systems.

The methods and associated tools allow to tackle control problems with state space of dimensions up to 7, and reachable state sets made of millions of elementary structures (typically, boxes or zonotopes, polyhedra). A major challenge is of course to design methods that scale with higher dimensional systems, and to face the well-known *curse of dimensionality* (see, e.g., [RMT13]). This problem is currently addressed via the design of new structures for representing union of convex sets such as "support functions" [LGG09] or "Brunowsky normal forms" [RMT13].

On the other hand, as pointed out in the introduction, practical consider-

ations specific to each case study often allow us to make realistic assumptions that considerably simplify the treatment of the problem. This was illustrated by the multilevel converter case study where, due to physical considerations, the number of admissible sequences of modes during an electrical cycle was only a small fraction of the total number of possible combinations.

General Conclusion

General Conclusion

Summary

In this thesis, we have presented how to compute schedulers in two different frameworks. In Part I, we have shown how the Inverse Method can be improved with a state space reduction technique introduced in Chapter 3. We have then used the Inverse Method to address both the problem of designing a scheduler in multiprocessor architectures and to give a robustness criterion, and how to find the schedulability region by using the Behavioral Cartography in Chapter 4. We have also shown the interest of our approach in Chapter 4 on several case studies from the literature and an industblablarial partner, and we have compared our approach to an analytic method.

In Part II, we have presented a novel approach to solve the problem of designing schedulers for affine sampled switched system using only forward computations. In Chapter 8, we have extended our approach to affine sampled switched systems and non-linear switched systems. In Chapter 9, we have applied our method to a real-life prototype provided by the SATIE Laboratory. We have also shown how it can be used to treat the problem of reachability analysis.

Future Research

For the schedulability analysis, it would be interesting to use model reduction before using the Inverse Method in order to scale up to real-life application. Indeed, our approach is exponential in the number of tasks, processors and jobs. A reduced model could help overcome the curse of dimensionality. The Behavioral Cartography could be improved to treat more efficiently this kind of case studies. So far, every integer point (or every point whose coordinates are a multiple of a fixed step) is tested in a sequential order. An interesting improvement would be to use theoretical results from the schedulability community that would allow to cover large zones as schedulable even if the Inverse Method did not cover it with a tile and better choose which next parameter valuation

should be investigated by the Inverse Method.

For the controllability of sampled switched systems, it would be interesting to have necessary and sufficient conditions for the location of controllable areas. So far, we only have a sufficient condition that is not always met in our case studies. The decomposition procedure could also be improved by having better splits than the dichotomous splits of the initial area that we are performing.

Extensions of our approach could be investigated. One of those is partial observability. In many case studies, some measures cannot be acquired. For example in power electronics, the value of the current can be tricky or even impossible to obtain. Controlling with partial observability and by estimating the missing values would be of prime interest for this kind of applications.

Another possible extension of our approach could be Partial Differential Equations (PDEs). As a long term goal, it would be interesting to be able to treat system governed by PDEs. As for now, many problems remain, for example, the large number of points of discretization, therefore a high dimensional state space, seems to forbid the use of our method. However, model reduction and state estimation could be investigated to overcome this difficulty.

It would be interesting to have controllers robust to change in the parameter valuation. So far, we have assumed that the coefficients in our equations are perfectly known (such as the capacitor, the resistor and the inductor values). As in the first part of this thesis, this is not the case in real-life systems. An interesting point would be to be able to provide robustness criterion to changes in the system (e.g., change in the value of a resistor, in the load of our system) or to select a control strategy that is the most resilient to those changes.

Bibliography

[AAM06] Y. Adbeddaïm, E. Asarin, and O. Maler. Scheduling with timed automata. *TCS*, 354(2):272–300, 2006.

[ABBL98] Luca Aceto, Patricia Bouyer, Augusto Burgueño, and Kim Guldstrand Larsen. The power of reachability testing for timed automata. In Vikraman Arvind and Ramaswamy Ramanujam, editors, *FSTTCS'98*, volume 1530 of *Lecture Notes in Computer Science*, pages 245–256. Springer, 1998.

[ABD+00] E Asarin, O Bournez, T Dang, O Maler, and A Pnueli. Effective Synthesis of Switching Controllers for Linear Systems. *Proceedings of the IEEE, Special Issue on Hybrid Systems*, 88(7):1011–1025, 2000.

[ABL98] Luca Aceto, Augusto Burgueño, and Kim G. Larsen. Model checking via reachability testing for timed automata. In Bernhard Steffen, editor, *TACAS 98*, volume 1384 of *Lecture Notes in Computer Science*, pages 263–280. Springer, 1998.

[ABSTDG12] M. Amin Ben Sassi, R. Testylier, T. Dang, and A. Girard. Reachability analysis of polynomial systems using linear programming relaxations. In *ATVA*, pages 137–151, 2012.

[ACD93] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[ACEF09] Étienne André, Thomas Chatain, Emmanuelle Encrenaz, and Laurent Fribourg. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science*, 20(5):819–836, 2009.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[ADM02] E. Asarin, T. Dang, and O. Maler. The d/dt tool for verification of hybrid systems. In *CAV*, pages 365–370, 2002.

[AF10] É. André and L. Fribourg. Behavioral cartography of timed automata. In *RP'10*, volume 6227 of *Lecture Notes in Computer Science*, pages 76–90. Springer, 2010.

[AFKS12] Étienne André, Laurent Fribourg, Ulrich Kühne, and Romain Soulat. IMITATOR 2.5: A tool for analyzing robustness in scheduling problems. In *FM'12*, volume 7436 of *Lecture Notes in Computer Science*, pages 33–36, Paris, France, 2012. Springer.

[AFS13] Étienne André, Laurent Fribourg, and Romain Soulat. Merge and conquer: State merging in parametric timed automata. In Dang-Van Hung and Mizuhito Ogawa, editors, *Proceedings of the 11th International Symposium on Automated Technology for Verification and Analysis (ATVA'13)*, volume 8172 of *Lecture Notes in Computer Science*, pages 381–396. Springer, October 2013.

[AHV93] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *STOC'93*, pages 592–601, New York, NY, USA, 1993. ACM.

[AK02] P.J. Antsaklis and X.D. Koutsoukos. Hybrid systems control. *Encyclopedia of Physical Sciences and Technology*, 7:445–458, 2002.

[AM02] Yasmina Adbeddaïm and Oded Maler. Preemptive job-shop scheduling using stopwatch automata. In *TACAS'02*, volume 2280 of *Lecture Notes in Computer Science*, pages 113–126. Springer-Verlag, 2002.

[And10a] Étienne André. *An Inverse Method for the Synthesis of Timing Parameters in Concurrent Systems*. Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, 2010.

[And10b] Étienne André. Synthesizing parametric constraints on various case studies using IMITATOR II. Research Report LSV-10-21, Laboratoire Spécification et Vérification, ENS Cachan, France, 2010.

[And13a] Étienne André. Observer patterns for real-time systems. In Yang Liu and Andrew Martin, editors, *18th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'13)*, pages 125–134. IEEE Computer Society, July 2013.

[And13b] Étienne André. Dynamic clock elimination in parametric timed automata. In *FSFMA*, volume 31 of *OpenAccess Series in Informatics*, pages 18–31. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, 2013.

[AS11] É. André and R. Soulat. Synthesis of timing parameters satisfying safety properties. In *RP'11*, volume 6945 of *Lecture Notes in Computer Science*. Springer, 2011.

[AS13] É. André and R. Soulat. *The Inverse Method*. Wiley-ISTE, jan 2013. 176 pages.

[ASB07] M. Althoff, O. Stursberg, and M. Buss. Reachability analysis of linear systems with uncertain parameters and inputs. In *In Proc. of the 46th IEEE Conference on Decision and Control*, pages 726–732, 2007.

[ASB08] M. Althoff, O. Stursberg, and M. Buss. Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In *CDC*, pages 4042–4048, 2008.

[ATS09] A. Abate, A. Tiwari, and S. Sastry. Box invariance in biologically-inspired dynamical systems. *Automatica*, 45(7):1601–1610, 2009.

[BB04a] Enrico Bini and Giorgio C. Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers*, 53(11):1462–1473, 2004.

[BB04b] Enrico Bini and Giorgio C. Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers*, 53(11):1462–1473, 2004.

[BBBB09] Christel Baier, Nathalie Bertrand, Patricia Bouyer, and Thomas Brihaye. When are timed automata determinizable? In *ICALP'09*, volume 5556 of *Lecture Notes in Computer Science*, pages 43–54, Rhodes, Greece, 2009. Springer.

[BCD+07] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime. Uppaal-tiga: Time for playing games! In *CAV*, pages 121–125, 2007.

[BDP10] O. Boudillet, D. Dalemagne, and T. Peron. Is integrated modular avionic a solution for ATV like spacecraft control. In *4th IAASS Conference*, Huntsville, Alabama, USA, 2010.

[Ber00] D.P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2nd edition, 2000.

[BHZ08] Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008.

[Bin04] Enrico Bini. *The Design Domain of Real-Time Systems*. PhD thesis, Scuola Superiore Sant'Anna, 2004.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[BL00] R.W. Brockett and D. Liberzon. Quantized feedback stabilization of linear systems. *Automatic Control, IEEE Transactions on*, 45(7):1279–1289, 2000.

[Bla99] F. Blanchini. Set invariance in control. *Automatica*, 35:1747–1767, 1999.

[BLV07] Reinder J. Bril, Johan J. Lukkien, and Wim F. J. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption revisited. In *ECRTS*, pages 269–279. IEEE Computer Society, 2007.

[BM83] Bernard Berthomieu and Miguel Menasche. An enumerative approach for analyzing time Petri nets. In *Proceedings of the IFIP 9th World Computer Congress*, pages 41–46. Elsevier Science Publishers, 1983.

[BPM05] A.G. Beccuti, G. Papafotiou, and M. Morari. Optimal control of the boost dc-dc converter. In *Proc. 44th IEEE Conference on Decision and Control European Control Conference (CDC-ECC '05)*, pages 4457 – 4462, dec. 2005.

[BRC05] J. Buisson, P.-Y. Richard, and H. Cormerais. On the stabilisation of switching electrical power converters. In *HSCC*, volume 3414 of *LNCS*, pages 184–197. Springer, 2005.

[Bro83] R. W. Brockett. Asymptotic stability and feedback stabilization. *Differential geometric control theory*, 27:181–191, 1983.

[BS95] J. A. Brzozowski and C. J. Seger. *Asynchronous Circuits*. Springer-Verlag, 1995.

[CC07] R. Clarisó and J. Cortadella. The octahedron abstract domain. *Science of Computer Programming*, 64(1):115–139, 2007.

[CGJ+00] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV'00*, pages 154–169. Springer-Verlag, 2000.

[CJL+09] F. Cassez, J. J. Jessen, K. G. Larsen, J.-F. Raskin, and P.-A. Reynier. Automatic synthesis of robust and optimal controllers - an industrial case study. In *HSCC*, pages 90–104, 2009.

[CPPMT09] I. Cervantes, F.J. Perez-Pinal, and A. Mendoza-Torres. Hybrid Control of DC-DC Power Converters. In *Renewable Energy (Chapter 10)*. T J Hammons, 2009.

[CPR08a] A. Cimatti, L. Palopoli, and Y. Ramadian. Symbolic computation of schedulability regions using parametric timed automata. In *RTSS*, pages 80–89, 2008.

[CPR08b] Alessandro Cimatti, Luigi Palopoli, and Yusi Ramadian. Symbolic computation of schedulability regions using parametric timed automata. In *RTSS'08*, pages 80–89. IEEE Computer Society, 2008.

[Dav05] Alexandre David. Merging DBMs efficiently. In *17th Nordic Workshop on Programming Theory*, pages 54–56. DIKU, University of Copenhagen, 2005.

[Dav06] Alexandre David. Uppaal DBM library programmer's reference. http://people.cs.aau.dk/~adavid/UDBM/manual-061023.pdf, 2006.

[Dil89] David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems 1989*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1989.

[DKRT97] Pedro R. D'Argenio, Joost-Pieter Katoen, Theo C. Ruys, and Jan Tretmans. The bounded retransmission protocol must be on time! In *TACAS*, volume 1217 of *LNCS*, pages 416–431. Springer, 1997.

[DLHT11] J. Ding, E. Li, H. Huang, and C. J. Tomlin. Reachability-based synthesis of feedback policies for motion planning under bounded disturbances. In *IEEE International Conference on Robotics and Automation (ICRA'11)*, pages 2160–2165, 2011.

[DTOC09] Z. Du, L.M. Tolbert, B. Ozpineci, and J.N. Chiasson. Fundamental frequency switching strategies of a seven-level hybrid cascaded h-bridge multilevel inverter. *IEEE Transactions on Power Electronics*, 24(1):25 –33, jan. 2009.

[FFL+12] G. Feld, L. Fribourg, D. Labrousse, B. Revol, and R. Soulat. Correct by design control of 5-level and 7-level converters. Research Report LSV-12-25, Laboratoire Spécification et Vérification, ENS Cachan, France, Dec 2012.

[Fin06] Olivier Finkel. Undecidable problems about timed automata. In *FORMATS'06*, volume 4202 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 2006.

[FK13] Laurent Fribourg and Ulrich Kühne. Parametric verification and test coverage for hybrid automata using the inverse method. *International Journal of Foundations of Computer Science*, 24(2):233–249, 2013.

[FKS13] Laurent Fribourg, Ulrich Kühne, and Romain Soulat. Constructing attractors of nonlinear dynamical systems by state space decomposition. In Christine Choppy and Jun Sun, editors, *Proceedings of the 1st French-Singaporean Workshop on Formal Methods and Applications (FSFMA'13)*, volume 31 of *Open Access Series in Informatics*, pages 53–60, Singapore, July 2013. Leibniz-Zentrum für Informatik.

[FLGD+11] 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 *CAV*, pages 379–395, 2011.

[FLMS12] Laurent Fribourg, David Lesens, Pierre Moro, and Romain Soulat. Robustness analysis for scheduling problems using the inverse method. In *TIME'12*, pages 73–80. IEEE Computer Society Press, 2012.

[Fre08] G. Frehse. PHAVer: algorithmic verification of hybrid systems past hytech. *STTT*, 10(3):263–279, 2008.

[FRL06] D. Flieller, P. Riedinger, and J.-P. Louis. Computation and stability of limit cycles in hybrid systems. *Nonlinear Analysis*, 64(2):352–367, 2006.

[FS13a] Laurent Fribourg and Romain Soulat. *Control of Switching Systems by Invariance Analysis: Application to Power Electronics*. Wiley-ISTE, July 2013. 144 pages.

[FS13b] Laurent Fribourg and Romain Soulat. Limit cycles of controlled switched systems: Existence, stability, sensitivity. In Laure Blanc-Féraud and Pierre-Yves Joubert, editors, *Proceedings of the 3rd International Workshop on New Computational Methods for Inverse Problems (NCMIP'13)*, volume 464 of *Journal of Physics: Conference Series*, Cachan, France, May 2013. IOS Press.

[FS13c] Laurent Fribourg and Romain Soulat. Stability controllers for sampled switched systems. In Parosh Aziz Abdulla and Igor Potapov, editors, *Proceedings of the 7th Workshop on Reachability Problems in Computational Models (RP'13)*, volume 8169 of *Lecture Notes in Computer Science*, pages 135–145, Uppsala, Sweden, September 2013. Springer.

[GHGGPGDM01] M. Gonzalez Harbour, J. J. Gutierrez Garcia, J. C. Palencia Gutierrez, and J. M. Drake Moyano. Mast: Modeling and analysis suite for real time applications. In *ECRTS*, pages 125–134, 2001.

[GHGPD01] M. González Harbour, J.J. Gutiérrez, J.C. Palencia, and J.M. Drake. MAST: Modeling and analysis suite for real-time applications. In *ECRTS*, 2001.

[Gir05] A. Girard. Reachability of uncertain linear systems using zonotopes. In *HSCC*, pages 291–305, 2005.

[Gir10] A. Girard. Synthesis using approximately bisimilar abstractions: state-feedback controllers for safety specifications. In *HSCC*, pages 111–120, 2010.

[Gir12] A. Girard. Low-complexity switching controllers for safety using symbolic models. In Maurice Heemels, Bart De Schutter, and Mircea Lazar, editors, *4th IFAC conference on Analysis and Design of Hybrid Systems, June, 2012*, Eindhoven, Pays-Bas, 2012.

[Gon03] J. M. Gonçalves. Region of stability for limit cycles of piecewise linear systems. In *IEEE Conference on Decision and Control*, 2003.

[GP05] A. Girard and G. J. Pappas. Approximation metrics for discrete and continuous systems. *IEEE Transactions on Automatic Control*, 52(5):782–798, 2005.

[GPM08] T. Geyer, G. Papafotiou, and M. Morari. Hybrid model predictive control of the step-down DC-DC converter. *IEEE Transactions on Control System Technology*, 16(6):1112 – 1124, jan. 2008.

[GPT10] A. Girard, G. Pola, and P. Tabuada. Approximately bisimilar symbolic models for incrementally stable switched systems. *IEEE Trans. on Automatic Control*, 55:116–126, 2010.

[Hen96] T. A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, LICS '96, pages 278–292, Washington, DC, USA, 1996. IEEE Computer Society.

[HHJ+05] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysis – the SymTA/S approach. *Computers and Digital Techniques, IEE Proceedings* -, 152(2):148 – 166, 2005.

[His01] I.A. Hiskens. Stability of limit cycles in hybrid systems. In *34th Annual Hawaii International Conference on System Sciences (HICSS-34)*. IEEE Computer Society, January 3-6, 2001.

[HRSV02] T.S. Hune, J.M.T. Romijn, M.I.A. Stoelinga, and F.W. Vaandrager. Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming*, 2002.

[JKDW01] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis*. Springer, 1 edition, September 2001.

[Küh98] W. Kühn. Zonotope dynamics in numerical quality control. *Mathematical Visualization*, pages 125–134, 1998.

[LGG09] C. Le Guernic and A. Girard. Reachability analysis of hybrid systems using support functions. In *CAV*, pages 540–554, 2009.

[Lib03] D. Liberzon. *Switching in Systems and Control*. Birkhausen, 2003.

[LL61] J. Lasalle and S. Lefschetz. *Stability by Lyapunov's Direct Method*. Academic Press, 1961.

[LM99] D. Liberzon and A.S. Morse. Basic problems in stability and design of switched systems. *Control Systems, IEEE*, 19(5):59–70, 1999.

[LPP+10] Thi Thieu Hoa Le, Luigi Palopoli, Roberto Passerone, Yusi Ramadian, and Alessandro Cimatti. Parametric analysis of distributed firm real-time systems: A case study. In *ETFA'10*, pages 1–8, 2010.

[LPPR13] Thi Thieu Hoa Le, Luigi Palopoli, Roberto Passerone, and Yusi Ramadian. Timed-automata based schedulability analysis for distributed firm real-time systems: a case study. *International Journal on Software Tools for Technology Transfer*, 15(3):211–228, 2013.

[LPY97] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

[LTS99] J. Lygeros, C. J. Tomlin, and S. Sastry. Controllers for reachability specifications for hybrid systems. *Automatica*, 35(3):349–370, 1999.

[MF92] T.A. Meynard and H. Foch. Multi-level conversion: high voltage choppers and voltage-source inverters. In *23rd Annual IEEE Power Electronics Specialists Conference*, pages 397 –403 vol.1, jun-3 jul 1992.

[MGS12] David Monchaux, Pascal Gast, and Jérémie Sangare. AvionicX: A demonstrator for the Next Generation Launcher Avionics. In *ERTS'12*, February 2012.

[min] MINIMATOR Web page. https://bitbucket.org/ukuehne/minimator/overview

[Mit07] I. M. Mitchell. Comparing forward and backward reachability as tools for safety analysis. In *HSCC*, pages 428–443, 2007.

[MP95] Oded Maler and Amir Pnueli. Timing analysis of asynchronous circuits using timed automata. In *CHARME'95*, pages 189–205, 1995.

[Oca13] Ocaml Team. OCaml Web page. http://caml.inria.fr/ocaml/index.fr.html, 2013.

[Oct13] Octave Team. Octave Web page. http://www.gnu.org/software/octave/, 2013.

[PB05] B. Picasso and A. Bicchi. Control synthesis for practical stabilization of quantized linear systems. *Rend. Sem. Mat. Univ. Pol. Torino, Control Theory and Stabil., I*, 63(4):397–410, 2005.

[PB08] B. Picasso and A. Bicchi. Hypercubes are minimal controlled invariants for discrete-time linear systems with quantized scalar input. *Nonlinear Analysis: Hybrid Systems*, 2(3):706 – 720, 2008.

[PG09] D.A. Patino Guevara. *Pilotage des cycles limites dans les systemès dynamiques hybrides. Application aux alimentations electriques statiques.* Ph.D Thesis, Nancy-université, 2009.

[PGH98] J. C. Palencia and M. Gonzalez Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *RTSS*, pages 26–37, 1998.

[PLE13] PLECS Team. PLECS Web page. http://www.plexim.com, 2013.

[PPL13] PPL Team. PPL Web page. http://bugseng.com/products/ppl/, 2013.

[RE02] K. Richter and R. Ernst. Event model interfaces for heterogeneous system analysis. In *DATE*, pages 506–513. IEEE Computer Society, 2002.

[RL00] M. Rubensson and B. Lennartson. Stability of limit cycles in hybrid systems using discrete-time lyapunov techniques. In *Proceedings of the 39th IEEE Conference on Decision and Control*, 2000.

[RMT13] M. Rungger, M. Mazo, and P. Tabuada. Specification-guided controller synthesis for linear systems and safe linear-time temporal logic. In *Proceedings of the 16th Intl. Conf. on Hybrid Systems: Computation and Control (HSCC)*, pages 333–342, 2013.

[RO98] J. Raisch and S.D. O'Young. Discrete approximation and supervisory control of continuous systems. *Automatic Control, IEEE Transactions on*, 43(4):569–573, 1998.

[RW89] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, January 1989.

[San93] S.R. Sanders. On limit cycles and the describing function method in periodically switched circuits. *IEEE Trans. Circuits and Systems*, 40(9):564–572, 1993.

[SBM06] Ramzi Ben Salah, Marius Bozga, and Oded Maler. On interleaving in timed automata. In *CONCUR*, volume 4137 of *LNCS*, pages 465–476. Springer, 2006.

[Sch86] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, Inc., New York, NY, USA, 1986.

[SEK03] M. Senesky, G. Eirea, and T.-J. Koo. Hybrid modelling and control of power electronics. In *HSCC*, pages 450–465, 2003.

[SG05] Z. Sun and S.S. Ge. *Switched Linear Systems. Control and Design*. Springer-Verlag, 2005.

[SGL97] Jun Sun, Mark K. Gardner, and Jane W.-S. Liu. Bounding completion times of jobs with arbitrary release times, variable execution times and resource sharing. *IEEE Transactions on Software Engineering*, 23(10):603–615, 1997.

[SHL+13] Romain Soulat, Guillaume Hérault, Denis Labrousse, Bertrand Revol, Gilles Feld, Stéphane Lefebvre, and Laurent Fribourg. Use of a full wave correct-by-design command to control a multilevel modular converter. In Philippe Lataire, editor, *Proceedings of the 15th European Conference on Power Electronics and Applications (EPE'13)*, Lille, France, September 2013. IEEE Power Electronics Society. To appear.

[SLS98] Danbing Seto, Dan P. Lehoczky, and Lui Sha. Task period selection and schedulability in real-time systems. In *RTSS*, 1998.

[Sou10] Romain Soulat. Améliorations algorithmiques d'un moteur de model-checking et études de cas. Rapport de Master, Master 2 Recherche Informatique Paris Sud 11, 2010.

[SSL+13a] Youcheng Sun, Romain Soulat, Giuseppe Lipari, Étienne André, and Laurent Fribourg. Parametric schedulability analysis of fixed priority real-time distributed systems. In Cyrille Artho and Peter Csaba Ölveczky, editors, *Preproceedings of the 2nd International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS'13)*, pages 179–194, Queenstown, New Zealand, October 2013.

[SSL+13b] Youcheng Sun, Romain Soulat, Giuseppe Lipari, Étienne André, and Laurent Fribourg. Parametric schedulability analysis of fixed priority real-time distributed systems. Research Report LSV-13-03, Laboratoire Spécification et Vérification, ENS Cachan, France, 2013.

[Tab05] P. Tabuada. Symbolic sub-systems and symbolic control of linear systems. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on*, pages 18–23, 2005.

[Tab08] P. Tabuada. An approximate simulation approach to symbolic control. *IEEE Trans. Automat. Contr.*, 53(6):1406–1418, 2008.

[Tab09] P. Tabuada. *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[Tar05] A. Tarantola. *Inverse Problem Theory and Methods for Model Parameter Estimation*. SIAM, 2005.

[Tea08] Simulink Team. Simulink Web page. http://www.mathworks.com/products/simulink/., 2008.

[TLS00] C.J. Tomlin, J. Lygeros, and S.S Sastry. A game-theoretic approach to controller design for hybrid systems. *Proceedings of the IEEE*, 88(7):949–970, 2000.

[WTVL06] Ernesto Wandeler, Lothar Thiele, Marcel Verhoef, and Paul Lieverse. System architecture evaluation using modular performance analysis: a case study. *International Journal on Software Tools for Technology Transfer*, 8(6):649–667, 2006.

[WY03] F. Wang and H.C. Yen. Timing parameter characterization of real-time systems. In *CIAA '03*, volume 2759 of *LNCS*, pages 23–34, 2003.

[Yov97] Sergio Yovine. Kronos: A verification tool for real-time systems. (kronos user's manual release 2.2). *International Journal on Software Tools for Technology Transfer*, 1:123–133, 1997.