PhD Dissertation

International Doctorate School in Information and Communication Technologies

FBK-IRST

DIT - University of Trento

Verification of Hybrid Systems using Satisfiability Modulo Theories

Sergio Mover

Advisor:

Dr. Alessandro Cimatti

FBK-IRST

Co-Advisor:

Dr. Stefano Tonetta

FBK-IRST

Abstract

Embedded systems *are formed by hardware and software components that interact with the physical environment and thus may be modeled as* Hybrid Systems*. Due to the complexity the system, there is an increasing need of automatic techniques to support the design phase, ensuring that a system behaves as expected in all the possible operating conditions.*

In this thesis, we propose novel techniques for the verification and the validation of hybrid systems using Satisfiability Modulo Theories *(SMT). SMT is an established technique that has been used successfully in many verification approaches, targeted for both hardware and software systems. The use of SMT to verify hybrid systems has been limited, due to the restricted support of complex continuous dynamics and the lack of scalability.*

The contribution of the thesis is twofold. First, we propose novel encoding techniques, which widen the applicability and improve the effectiveness of the SMT-based approaches. Second, we propose novel SMT-based algorithms that improve the performance of the existing state of the art approaches. In particular we show algorithms to solve problems such as invariant verification, scenario verification and parameter synthesis. The algorithms fully exploit the underlying structure of a network of hybrid systems and the functionalities of modern SMT-solvers.

We show and discuss the effectiveness of the the proposed techniques when applied to benchmarks from the hybrid systems domain.

Keywords

[Formal Verification, Hybrid Systems, Cyber-Physical Systems, Hybrid Automata, Model Checking, Satisfiability Modulo Theory, Scenario Verification, Parameter Synthesis]

Contents

List of Tables

List of Figures

Acknowledgements

First of all, I want to thank my advisors, Alessandro Cimatti and Stefano Tonetta, for all the help they gave me during the doctorate. They were always available for discussions and to give me useful insights. I am grateful for all the efforts they made and for all the time they dedicated to me. This thesis would have not been possible without their help.

Then, I would like to thank Dr. Ashish Tiwari for all the insights and the inputs he gave me. I also thank him for hosting me at SRI International, it was a great opportunity for me.

I have to thank Alberto Griggio, for his support for using MathSAT and IC3. All these works were a key enabler for the research presented in this thesis.

I want to thank all the friends that helped me during the Ph.D. First of all, my close friends in FBK, Andrea Micheli, Cristian Mattarei, Marco Gario, Alessandro Mariotti, Marco Roveri, Gianni Zampedri, Alberto Griggio, Andrea Avancini, Bas Schaasfma, Mirko Sessa, Michele Dorigatti, Ilaria Sambarino, Mirko Sessa, Benjamin Bittner. They always encouraged me and supported me during hard times. Moreover, they made the life at work a huge fun! Then, I would like to thank the friends I met at SRI, Adria Gascon, Jan Leike, Muhammad Rizwan Asghar.

I would like to thank several people for their feedback and useful discussions: Dr. Marco Bozzano, Roberto Cavada, Dr. Viktor Schuppan, Dr. Iman Narasamdya, Dr. Bruno Dutertre, Dr. John Rushby, Dr. Natarajan

Shankar, Dr. Sam Owre.

Of course, I have to thank my friends outside academia, Matteo, Tiziano, Romina in Trento and Alessandro, Luis and Emma while staying in the U.S..

Finally, I have to thank my parents and my brother Andrea for all their patience and support.

Chapter 1

Introduction

1.1 Motivations

Embedded systems consist of several software and hardware components that interact with the physical environment. These kinds of systems are increasingly used in many industrial sectors, such as automotive, aerospace, consumer electronic, medical and manufacturing. Paradigmatic examples of embedded systems are the control software of airplanes (e.g. the auto-pilot software), the anti-breaking system (ABS) of cars, ... One key feature of these systems is that discrete computations (e.g. the control software) interact with the physical environment (e.g. the velocity of an object). Then, real embedded systems usually consist of different components that interact, for example exchanging messages, and that may evolve asynchronously.

The failure of embedded systems is not desirable, since they are often used in safety critical applications (e.g. the control software of an airplane), where a failure may have severe consequences and very high costs. To reduce such costs, it is important to identify and fix the bugs in the system in the early stages of the design flow. Due to the complexity of the systems, there is an increasing need of automatic techniques that support the design phase and allow the designer to validate the system and to certify that it works as expected.

Formal verification provides several techniques that help to identify bugs in the early stages of the design phase. In *Formal verification* a system is modeled using formal languages (e.g. mathematical logic) and is analyzed by means of rigorous techniques. Examples of formal verification techniques are *model checking*, *theorem proving*, *abstract interpretation*. Compared to the common practices used to find bugs, like *testing* and *simulation*, formal verification ensures the correctness of the system with respect to a specification for all the possible inputs. Well known problems in formal verification are the reachability problem and its dual, the invariant verification problem.

The *Hybrid systems* formalism allow to represent systems that exhibit both discrete behaviors, where computations are supposed to happen instantaneously, and continuous behaviors, where physical quantities evolve continuously in time. Thus, they are a suitable formalism to model embedded systems.

The application of formal verification techniques to hybrid systems pose several challenges. First, the analysis of such systems is difficult due to the interaction of the discrete and the continuous behaviors of the system. On the one hand, the analysis algorithm must take into account the discrete state space of the system and its transitions. On the other hand, the algorithm must reason on the differential equations that describe the evolution of the physical quantities of the system. Then, the verification algorithms may not be efficient enough to analyze complex systems, where multiple components interact. In fact, the number of components increase the state-space of the system, and thus complicate the verification tasks. In the literature there have been several approaches to tackle verification problems for hybrid systems (See [Alu11] for a survey). The prominent line of research develops methods based on the computation of the reachable states. Instead, other families of approaches are either based on deductive verification or on abstraction techniques.

Several verification techniques for hybrid systems [dMRS03, ABCS05, ÁBKS05] are based on *Satisfiability Modulo Theories* (SMT) [BSST09]. SMT is an established paradigm in formal verification. The SMT problem consists to check the satisfiability of first-order logic formulas interpreted with respect to a background theory. An example of theory is the *Theory of Reals*, which formalizes the arithmetic operations on the real numbers. SMT-based verification techniques have been successfully used to analyze both hardware and software systems. With respect to hybrid systems, it is possible to *encode* both the discrete and the continuous behavior of an hybrid system in a symbolic transition systems, expressed as SMT formulas. Then, the symbolic transition system can be analyzed using the SMT-based verification algorithms.

However, the current state of the art in the verification of hybrid systems using SMT solvers presents important weaknesses. First, the applicability of the SMT-based techniques is limited to hybrid systems with simple dynamics. This limits the significance of the approaches based on SMT, which cannot be used on several real life examples. Second, the performance of the SMT-based algorithms is not satisfactory in several contexts, in particular in the case of distributed hybrid systems. This constitutes a problem when verifying embedded systems, which are often designed as distributed components.

1.2 Contribution of the thesis

In this thesis we investigate the verification problem of hybrid systems using Satisfiability Modulo Theories.

We rely on the *Hybrid Automata Network* framework [Hen96] (HAN) to formalize hybrid systems. While an hybrid automaton models a single hybrid system, a network of hybrid automata models a set of hybrid systems that interact using a synchronization mechanism.

Our contributions can be categorized in two broad classes:

1. We provide novel *encodings* of hybrid automata and hybrid automata network.

   First, we investigate a novel encoding technique that handles *Non-linear Hybrid Automata*, while state of the art encodings only deal with *Linear Hybrid Automata*. This extends the applicability of SMT-based verification algorithms. Then, we investigate the encodings of hybrid automata networks, exploiting alternatives semantics [BJLY98]. Finally, we improve the precision of the existing encoding techniques based on abstraction for the class of *Linear Hybrid Systems*.

2. We investigate novel algorithms to solve different verification problems for networks of hybrid automata.

   We investigate on novel and efficient algorithms that exploit the structure of the hybrid automata network for both the reachability and the scenario verification problems. While standard techniques reason on the composition of the different automata in the network, we propose a technique that exploits an alternative semantic, called "shallow synchronization". The novel approaches are more efficient.

   Then, we provide two novel algorithms that works for infinite-state transition systems, which can be directly applied to the encoding of a hybrid automata network. One algorithm solve the reachability problem, while the other the parameter synthesis problem.

In details, the contributions of this thesis can be summarized as follows.

1. *We extend the applicability of SMT-based algorithms to hybrid systems with richer dynamics [CMT12, CMT13b]*

   The state of the art in the precise encoding of hybrid systems in SMT formulas is limited to the simple class of *Linear Hybrid Automata*. A subset of the more expressive classes of non-linear hybrid automata may be encoded in a first-order logic formula, requiring universal quantifiers to encode the invariant condition. In several cases (e.g. when there are transcendental functions) the quantifier cannot be eliminated (the problem is not decidable) while in other cases (e.g. non-linear real arithmetic) the quantifier cannot be removed efficiently. This issue hinders the applicability of SMT algorithms, which requires quantifier-free formulas. We propose a novel technique to obtain a quantifier-free encoding for several classes of hybrid systems, we apply the technique to several examples and we characterize the sub-classes of systems that can be handled automatically. The encoding extends the applicability of the SMT-based algorithm to a richer class of systems.

2. *We improve relational abstraction techniques for Linear Hybrid Systems [MCTT13].*

   Relational abstraction is a technique that abstracts the dynamic of a hybrid system in a SMT formula. The generated abstraction can be verified using SMT-based algorithms. The shortcomings of of the existing relational abstraction techniques for *Linear Hybrid Systems* are that the produced abstractions may be too coarse (to prove an invariant property) and that the precision of the abstraction cannot be increased. We improve the existing technique that generates relational abstractions for *Linear Hybrid Systems*, allowing it to create more precise abstractions. We show that the increased precision of the abstraction is needed to prove properties of models of practical interest.

3. *We study novel SMT-based algorithms for the reachability problem of network of hybrid systems [BCL$^+$10a].*

   First, we investigate a novel *Bounded Model Checking* (BMC) encoding, based on the *Shallow Synchronization Semantic*, which are specific for a network of hybrid automata. The encoding exploits the locality of the automata in the network. Compared to the existing approaches, where the BMC is not aware of the structure of the network, the new encoding shows better performance if the path to the target set of states does not involve the interaction of all the components of the network.

   Then, we present a verification algorithms for infinite-state transition systems based on an abstraction refinement loop. The approach exploits an existing technique that embeds predicate abstraction and k-induction.

4. *We investigate efficient algorithms to prove the feasibility of a scenario specification for a network of hybrid automata [CMT11a, CMT11c, CMT13c].*

   We provide two algorithms based on SMT to solve the scenario verification problem. Both algorithms exploit the shallow synchronization semantic. The algorithms advance the state of the art, demonstrating a dramatic improvement in the running time with respect to the existing approaches. Then, another contribution is an effective technique to provide debug information in the case a scenario does not hold.

5. *We propose a novel and efficient parameter synthesis algorithm for infinite-state transition systems [CGMT13].*

We propose a novel algorithm that solves the parameter synthesis problem. The algorithm works for infinite-state transition systems and thus can be applied to the hybrid systems considered in this thesis. The algorithm shows very good performance compared to the state of the art.

1.3 Structure of the thesis

The thesis is divided in four parts.

The first part introduce the background notions on SMT and the formalism of hybrid automata network. It also describes the state-of-the art SMT-based verification techniques.

The second part describes the contributions related to the encoding of hybrid systems:

- Chapter 3 describes the encoding of a hybrid automaton in a symbolic transition systems. The chapter describes the encoding approach for non-linear hybrid automata.
- Chapter 4 presents the global-time and the local-time encodings of a hybrid automata network.
- Chapter 5 describes the improved relational abstraction techniques for linear hybrid systems.

The third part contains the description of the novel verification algorithms.

- Chapter 6 describes the verification algorithms that solve the reachability problem: Bounded Model Checking using Shallow Synchronization (Section 6.2) and K-induction with implicit predicate abstraction (Section 6.3).

- Chapter 7 describes the algorithms for the scenario verification problem.
- Chapter 8 presents a novel algorithm for the parameter synthesis problem of infinite-state transition systems.

The fourth part of the thesis presents the tool and the experimental evaluation:

- Chapter 9 describes the HyComp tool, which implements most of the techniques presented in the thesis.
- Chapter 10 describes the experiments performed on the techniques presented in the thesis and discusses the obtained results.

Finally, Chapter 11 concludes the thesis and describes some future works.

Part I

Background notions

Chapter 2

Background

2.1 Satisfiability Modulo Theory

2.1.1 The Satisfiability Modulo Theory Problem

Our setting is standard first-order logic. Let Σ be a first-order *signature* containing predicates and function symbols with their arity and $\mathcal{V}$ be a set of variables. A 0-ary predicate symbol A is called a *Boolean atom* while a 0-ary function symbol c is called a *constant.* A Σ-*term* is either a variable or it is built applying function symbols in Σ to Σ-terms. If p is a predicate with arity n and $t_1, \ldots, t_n$ are Σ-terms, then $p(t_1, \ldots, t_n)$ is a Σ-*atom.* A Σ-*formula* is built in the usual way with the universal and existential quantifiers, $\exists$, $\forall$, the Boolean connectives $\wedge, \vee, \neg$ and Σ-atom. We will use the standard abbreviations for the other Boolean operators, "$\phi_1 \rightarrow \phi_2$" for "$\neg\phi_1 \vee \phi_2$" and "$\phi_1 \leftrightarrow \phi_2$" for "$(\phi_1 \rightarrow \phi_2) \wedge (\phi_2 \rightarrow \phi_1)$". A first-order Σ-*theory* $\mathcal{T}$ is a set of first-order sentences with signature Σ, where a sentence is a Σ-formula without free variables. We assume that the symbols $=$, $\bot$, and $\top$ are part of the language, even if they are not explicitly contained in the signature, and are interpreted as the identity, false, and true, respectively.

We assume the standard first-order notion of interpretation, satisfiability, validity and logical consequence. We write $\Gamma \models \phi$ to denote that the formula ϕ is a logical consequence of all the formulas in the (possibly infinite) set Γ.

A Σ-*structure* $\mathcal{M}$ is a model of a Σ-theory $\mathcal{T}$ if $\mathcal{M}$ satisfies every sentence in $\mathcal{T}$. A Σ-formula is satisfiable in $\mathcal{T}$ ($\mathcal{T}$-satisfiable) if it is satisfiable in a model of $\mathcal{T}$. A Σ-formula is valid in $\mathcal{T}$ ($\mathcal{T}$-satisfiable) if it is satisfiable in all models of $\mathcal{T}$. We write $\Gamma \models_{\mathcal{T}} \phi$ to denote $\Gamma \cup \mathcal{T} \models \phi$. Two Σ-formula ϕ_1 and ϕ_2 are $\mathcal{T}$-*equisatisfiable* if and only if ϕ_1 is $\mathcal{T}$-satisfiable if and only if ϕ_2 is $\mathcal{T}$-satisfiable.

The *Satisfiability Modulo Theory* problem (SMT ($\mathcal{T}$)) is the problem of checking if a Σ-formula ϕ is satisfiable, for some background theory $\mathcal{T}$.

2.1.2 Theories of interest

We are mainly interested in the theories over reals and rational numbers. For brevity, we do not discuss other theories, and we do not give the axioms that describe the theories. A thorough exposition on the topic may be found in [BM07a].

We will mostly consider the *Theory of Rationals*, $\mathcal{T}(\mathbb{Q})$ and we will also call it *Linear Arithmetic over Rationals.* The signature of the theory is $\Sigma_{\mathbb{Q}} = \{0, 1, +, -, =, \geq\}$ where the constants 0 and 1 are interpreted as rational numbers, and the binary function symbols and predicate symbols $+, =, \geq$ are interpreted with the corresponding operations and relations over the rationals. The others comparison operators, $<, >, \leq, \neq$ may be obtained from $\Sigma_{\mathbb{Q}}$ (i.e. $a < 0$ as $\neg(a \geq 0)$, $a \leq 0$ as $\neg(a > 0)$, $a \neq 0$ as $\neg(a = 0)$). As an intuition, the resulting language consists of quantifier-free Boolean combinations of atoms in the form $\sum a_i \cdot x_j \bowtie a$, where x_j is a variable, $a_i, a \in \mathbb{Q}$ and $\bowtie \in \{<, \leq, >, \geq, \neq\}$.

We will call *linear predicates* all the Σ-*atom* constructed from the signature $\Sigma_{\mathbb{Q}}$.

Example 1 *The formula* $x < y \wedge (x + 3 = z \vee z > y)$ *is satisfiable in the theory of* $\mathcal{T}(\mathbb{Q})$*, since* $x := 5, y := 6, z := 8$ *is a model for the formula.*

The *Theory of Reals*, $\mathcal{T}(\mathbb{R})$, (also called *elementary algebra*) has the signature $\Sigma_{\mathbb{R}} = \{0, 1, +, -, \cdot, =, \geq\}$, where 0 and 1 are the real numbers constants, $+, -, \cdot$ are the usual addition, subtraction and multiplication operators and $=, \geq$ are the usual equal and greater than or equal operators.

Note that the satisfiability of both theories is decidable. For $\mathcal{T}(\mathbb{Q})$, satisfiability is decidable through well know algorithms (e.g. the Simplex algorithm or Fourier-Motzkin elimination [DE73]), while $\mathcal{T}(\mathbb{R})$ may be decided using *Cylindrical Algebraic Decomposition* (CAD) [Col75].

In the most general case, we will consider an extension of the theory of reals, $\mathcal{T}(\overline{\mathbb{R}})$, such that its signature $\Sigma_{\overline{\mathbb{R}}}$ extends the structure $\Sigma_{\mathbb{R}}$ with transcendental functions such as the exponential and the trigonometric functions. The transcendental functions are unary interpreted function symbols, and are interpreted with their standard semantics. Note that the satisfiability of this theory is not decidable.

2.1.3 SMT solvers

SMT solvers are tools which implement decision procedures for the SMT problem. The most efficient implementations of SMT solvers use the so-called "lazy approach", where a SAT solver is tightly integrated with a $\mathcal{T}$-solver. The role of the SAT solver is to enumerate the truth assignments to the Boolean abstraction of the first-order formula. The Boolean abstraction has the same Boolean structure of the first-order formula, but "replaces" the predicates that contain $\mathcal{T}$information with fresh Boolean variables. The Boolean abstraction of the Example 1 is $a \wedge (b \vee c)$, where a, b, c are fresh Boolean variables. The $\mathcal{T}$-solver is invoked when the SAT solver finds a model for the Boolean abstraction: the Boolean model maps directly to a conjunction of $\mathcal{T}$atoms, which the $\mathcal{T}$-solver can handle. If the conjunction is satisfiable also the original formula is satisfiable. Otherwise the $\mathcal{T}$-solver returns a conflict set which identifies a reason for the unsatisfiability. Then, the negation of the conflict set is learned by the SAT solver in order to prune the search. Examples of solvers based on the "lazy approach" are MathSAT [CGSS13], Z3 [dMB08], Yices [DdM06] and OpemSMT [BPST10].

SMT-solvers often construct models in the case a formula is satisfiable and proofs if it is unsatisfiable. Proofs are used to generate additional information, such as *unsatisfiable cores* and *interpolants*.

We recall the standard notion of *conjunctive normal form* (CNF) for a formula ψ. A *literal* is a Σ-atom or the negation of a Σ-atom. A clause is a disjunction of literals. A formula is in *conjunctive normal form* (CNF) if it is a conjunction of clauses. It is well known that given a non-CNF formula we can obtain an equi-satisfiable CNF formula in polynomial time.

The *unsatisfiable core* for an unsatisfiable CNF formula ϕ is a formula ψ such that ψ is unsatisfiable and $\phi = \psi \wedge \psi'$, for a (possibly empty) formula ψ' (See [CGS11] for a survey on the computation of unsatisfiable core).

Given two formulas ϕ and ψ, with $\phi \wedge \psi \models \bot$, the *Craig Interpolant* (from now on only interpolant) of $\phi \wedge \psi$ is a formula I such that $\models \phi \rightarrow I$, $\psi \wedge I \models \bot$, and every uninterpreted symbol of I occurs both in ϕ and ψ. Intuitively, the interpolant is an over-approximation of ϕ "guided" by ψ (for interpolant computation, See [CGS10]).

Most modern SMT solvers also feature an *incremental* interface, i.e. they are able to tackle sequences of satisfiability problems efficiently, by reusing theory information discovered during the previous searches.

We assume that the sequences of problems have the following form, where each problem may inherit from the preceding one the variables and a substantial subset of sub-formulas:

$\gamma(0) \wedge \beta(0)$
$\gamma(0) \wedge \gamma(1) \wedge \beta(1)$
$\gamma(0) \wedge \gamma(1) \wedge \gamma(2) \wedge \beta(2)$
...

The non-monotonicity of the encoding is handled with a standard stack-based interface of the SMT solver (Push, Assert, Solve, Pop primitives). This allows, after asserting $\gamma(k)$, to set a backtrack point (Push), assert $\beta(k)$ (Assert), check the satisfiability of the conjunction of the asserted formulas (Solve), and to restore the state of the solver (i.e. asserted formulas and learned clauses) at the backtrack point (Pop). This way, the k+1-th problem is solved keeping all the learned clauses related to $\gamma(0), \ldots, \gamma(k)$.

The quantifier-elimination problem consists to compute a quantifier-free formula that is equivalent to a formula with quantifiers [BM07a]. A theory admits quantifier elimination if there exists an algorithm that solve the quantifier elimination problem. SMT solvers have also been extended to solve the quantifier elimination problem. For example, for the *Theory of Rationals*, the SMT solver is used as an efficient enumerator of models [Mon10]. Then, the approach relies on standard quantifier elimination algorithm for linear arithmetic [DE73, LW93].

2.2 First-order Transition Systems

We will represent transition systems symbolically using first-order formulas.

Given a first-order signature Σ with variables $\mathcal{V}$, and a natural number $i \in \mathbb{N}_{\geq 0}$, we denote with Σ', $\dot{\Sigma}$, Σ^i the signatures obtained by replacing each symbol s in Σ with $s', \dot{s}$ and s^i respectively. Similarly, given a set V, we denote with $V', \dot{V}, V^i$ the copies of such set such that $V' = \{v' \mid v \in V\}$,

$\dot{V} = \{\dot{v} \mid v \in V\}$, $V^i = \{v^i \mid v \in V\}$. Finally, we denote with $\mathcal{A}$ the set of Boolean atoms of the signature Σ with variables $\mathcal{V}$ (i.e. $\mathcal{A} = \{A \mid A \in \Sigma,\ \text{A is a 0-arity predicate}\}$).

Definition 1 (First-order Transition System) *Given a first-order signature Σ with variables $\mathcal{V}$, we denote with $\Sigma', \dot{\Sigma}$,*

Let Σ be a first-order signature, $\mathcal{V}$ a set of variables and let denote with $\mathcal{A} = \{A \mid A \in \Sigma\ A \text{ is a 0-arity predicate}\}$ the set of Boolean atoms of Σ. A first-order Σ-Transition System is a tuple $S = \langle V, \mathcal{W}, Init, Inv, Trans\rangle$ such that:

- *$V \subseteq (\mathcal{V} \cup \mathcal{A})$ is a set of state variables,*
- *$\mathcal{W} \subseteq (\mathcal{V} \cup \mathcal{A})$ is a set of input variables,*
- *$V \cap \mathcal{W} = \emptyset$, $V \cup \mathcal{W} = (\mathcal{V} \cup \mathcal{A})$,*
- *$Init$ is a first-order Σ-formula over V (called initial condition);*
- *Inv is a first-order Σ-formula over V (called invariant condition);*
- *$Trans$ is a first-order Σ-formula over $V \cup \mathcal{W} \cup V'$ (called transition condition).*

An assignment s to the variables V is a *state* of the transition system, while an assignment a to the variables $\mathcal{W}$ is an *input*. We denote with $s', \dot{s}, s^0, s^1, \ldots$ the corresponding assignment to the copy $V', \dot{V}, V^0, V^1, \ldots$ of V. Given a set of variables L and an assignment s, we denote with $s_{|L}$ the assignment s restricted to the variables in L. $s_{|L}$ contains only the assignment corresponding to the variables in L. If x is a variable, we will use the shorthand $s(x)$ to refer to the value of x in the assignment s (e.g. if $s = \langle x = 1\rangle$, $s(x) = 1$). Finally, given the assignments s_1, s_2 defined respectively on the set of variables X_1, X_2, with X_1 and X_2 disjoints (i.e.

$X_1 \cap X_2 = \emptyset$), we denote with $\langle s_1, s_2\rangle$ the assignment to the variables $X_1 \cup X_2$ obtained from s_1 and s_2.

Definition 2 (Path) *A sequence $\pi = s_0; a_1, s_1; \ldots; a_k; s_k$ of states and inputs is a model (also called* path*) of the transition system $S = \langle V, \mathcal{W}, Init, Inv, Trans\rangle$ iff:*

- *s_0 satisfies $Init$;*
- *for every $0 \leq i \leq k$, s_i satisfies Inv;*
- *for every $0 \leq i < k$, s_i, a_{i+1}, s_{i+1} satisfy $Trans$.*

Remark 1 *In order to keep the notation as simple as possible, the above definition does not distinguish symbols that are rigid and interpreted by $\mathcal{T}$ from the actual variables of the system. We assume that such symbols do not occur as "primed" in the transition condition although their interpretation is the same in all the states of a sequence.*

We say that a state s is *reachable* if there exists a path $\pi = s_0; a_1; s_1; \ldots; a_k; s_k$ such that $s_k = s$.

Definition 3 (Trace) *A* trace *is a sequence of events $w = a_1; \ldots; a_k$, where a_i is an assignments to variables in $\mathcal{W}$.*

Given $A' \subseteq A$, the projection $w_{|A'}$ of w on A' is the sub-trace of w obtained by removing all events in w that are not in A'.

To simplify the exposition, in particular to describe some reachability algorithms and when we are not interested in the traces of the system, we will use a more concise definition of first-order transition system, which does not have the invariant formula and input variables (i.e. $S = \langle V, Init, Trans\rangle$). We will specify when we use this definition instead of the one with input variables and the invariant formula. Note that this does not restrict the expressivity of the formalism. Input variables can be encoded as state variables, under the assumption that they do not compare in state formulas (i.e. they do not compare in the $Init$ and Inv formulas). The Inv formula can be encoded in the $Trans$ formula. This is obtained conjoining the $Trans$ formula with the Inv formula and its primed version Inv'.

Definition 4 (Parallel composition) *The parallel composition $S_1||S_2$ of two symbolic transition systems $S_1 = \langle V_1, \mathcal{W}_1, Init_1, Inv_1, Trans_1\rangle$ and $S_2 = \langle V_2, \mathcal{W}_2, Init_2, Inv_2, Trans_2\rangle$, is the symbolic transition system $\langle V, \mathcal{W}, Init, Inv, Trans\rangle$:*

- *$V = V_1 \cup V_2$;*
- *$\mathcal{W} = \mathcal{W}_1 \cup \mathcal{W}_2$;*
- *$Init = Init_1 \wedge Init_2$;*
- *$Inv = Inv_1 \wedge Inv_2$;*
- *$Trans = Trans_1 \wedge Trans_2$.*

2.3 SMT-based verification

In this section we overview the existing verification techniques for first-order transition systems using SMT solvers. In particular, we focus on the verification of invariant properties:

Definition 5 (Safety) *Let $S = \langle V, \mathcal{W}, Init, Inv, Trans\rangle$ be a first-order transition system and $P(V)$ a Σ-formula over the state variables V, also called invariant property. $S \models P$ iff there are no paths of S that can reach a state in P. In this case we say that S is safe.*

2.3.1 Verification Algorithms

Notation

Given a formula $\psi(V)$ over the variables V, we denote with $\psi(V^i)$ (or simply ψ^i when V is clear from the context) the formula obtained by replacing all the variables $v \in V$ with v^i. Similarly, given a formula $\psi(V, \mathcal{W}, V')$ over the variables $V, \mathcal{W}, V'$, we denote with ψ^i the formula obtained replacing all the variables $v \in V$ with v^i, $w \in \mathcal{W}$ with w^i, and $v' \in V'$ with v^{i+1}.

Bounded Model Checking

Bounded Model Checking (BMC) [BCCZ99] is a symbolic reachability technique that explores all the paths of the system from the initial state up to a fixed length. The technique was first introduced for finite-state transition systems [BCCZ99] to find violations to *Linear Temporal Logic (LTL)* properties [Pnu77]. The technique was extended to infinite-state transition systems [1] in [dMRS02], substituting the underlying decision procedure, a SAT solver, with an SMT solver.

Given a FOTS $S = \langle V, \mathcal{W}, Init, Inv, Trans \rangle$ and a set of target states $\neg P(V)$ over the variables in V, the BMC problem consists of determining if $\neg P$ is reachable in S in k steps (i.e. if there is a path $\pi = s_0; a_1; s_1; \ldots; a_k; s_k$ in S such that $s_k \models \neg P$). We encode the BMC problem as follows:

$$path(k) := Init^0 \wedge \bigwedge_{i=0}^{k} Inv^i \wedge \bigwedge_{i=0}^{k-1} Trans^i, \tag{2.1}$$

$$BMC(k) := path(k) \wedge \neg P^k \tag{2.2}$$

The formula $BMC(k)$ is satisfiable iff there exists a path from the initial state of S to a state in $\neg P$ of length k. An SMT solver is used to query

[1]While we present BMC to check invariant properties, the authors of [dMRS02] shows an encoding for LTL properties.

for the satisfiability of $BMC(k)$. The $BMC(k)$ formula may be slightly modified to explore all the paths of length *at least* k as follows:

$$BMC({}_{\leq}k) := path(k) \wedge \bigvee_{i=0}^{i \leq k} \neg P^i \tag{2.3}$$

We can exploit the incremental feature of the SMT solver to look for a k such that the formula $BMC(k)$ is satisfiable. The problem is presented to the solver in the following form: $\gamma(0) := Init^0 \wedge Inv^0$, $\gamma(i) := Trans^{i-1} \wedge Inv^i$, for $i > 0$, and $\beta(i) := \neg P^i$, for $i \geq 0$.

K-Induction

K-induction [SSS00] is a technique that aims to prove that a safety property $P(V)$ holds in a transition system S. By exploiting a reasoning similar to the induction principle, the technique consists of finding a bound k such that both a *base* step and an *inductive* step hold.

The base step consists of proving that P holds for the first k steps of the system. This can be done checking the unsatisfiability of $BMC(i)$, for $0 \leq i \leq k$. Then, the inductive step hold if either one of the *forward* or the *backward* steps hold. The forward step consists to check if any new state is reachable by a path of length $k+1$. If it is not the case, since all the paths up to k cannot reach a violation to P, then the algorithm explored *all* the paths of the systems, proving that $S \models P$. The backward step consists to check if there exists a loop-free path of length $k+1$ where P holds for k steps and reaches $\neg P$.

The forward step holds if the following formula is unsatisfiable:

$$kindfw(k) := path(k+1) \wedge simple(k+1) \tag{2.4}$$

$$simple(k) := \bigwedge_{0 \leq i < j \leq k} \neg \bigwedge_{v \in V} v^i = v^j \tag{2.5}$$

where *simple*(k) encodes that all the states in a path are different. In practice, *simple*(k) encodes a loop-free path, which is also called *simple path*.

The backward step holds if the following formula is unsatisfiable:

$$kindbw(k) := \bigwedge_{i=0}^{k} Inv^i \wedge \bigwedge_{i=0}^{k-1} Trans^i \wedge simple(k+1) \wedge \bigwedge_{i=0}^{i\leq k}(P^k) \wedge \neg P^{k+1} \tag{2.6}$$

Efficient algorithms that exploit the solver incrementality to interleave the step and forward satisfiability checks have been presented in [ES03].

As for BMC, k-induction may be applied to infinite-state transition systems [dMRS03, Pik07, SDS08, SD10]. However, in this case the approach may not be effective since the step case requires to find a loop-free path. Since the domain of the variables V of the system is infinite, the system may have an *infinite number of paths*, and thus *kindbw* and $kindfw$ will always be satisfiable.

In [Pik07, SDS08, SD10] the authors overcome the issue by manually strengthening the inductive check with additional lemmas. The technique consists of proving additional lemmas on the transition system S, and then add them to the invariant conditions of S. The right lemmas to add can be found looking at the counter-example to induction (i.e. a path that shows a witness of failure for the inductive step). Intuitively, the lemmas should be able to prove that the states in this path are not reachable. Also the lemmas should be proved to hold in S, for example applying again k-induction. The drawbacks of the approach are that it requires a manual intervention, a deep knowledge of the system and several iterations.

Another technique [Ton09] tries to solve the problem combining k-induction with abstraction techniques. We present the details of the approach in 6.3.

Interpolation-based Model Checking

Interpolation-based model checking [McM03, McM05] exploits the notion of *Craig Interpolants* to approximate the reachable states of the systems. The problem is formulated dividing the formula that encodes a set of paths of length k in a prefix of length $j < k$ and a suffix of length $k - j$.

$$\mathit{Pref}(j) := \mathit{Init}^0 \wedge (\bigwedge_{i=0}^{j} \mathit{Inv}^i) \wedge (\bigwedge_{i=0}^{j-1} \mathit{Trans}^i) \tag{2.7}$$

$$\mathit{Suff}(k-j) := (\bigwedge_{i=j}^{k} \mathit{Inv}^i) \wedge (\bigwedge_{i=j}^{k-1} \mathit{Trans}^i) \wedge (\bigvee_{i=j}^{k} \neg P^i) \tag{2.8}$$

Note that $\mathit{Pref}(j) \wedge \mathit{Suff}(k-j)$ encodes all paths of length k checking for the existence of a bug in all the states s_i such that $j \leq i \leq k$.

The interpolation-based algorithm tries to build an inductive invariant R that prove that $S \models P$. A formula R is an inductive invariant for the transition system S and the property P if:

1. $I \models R$,
2. $R \wedge T \wedge R'$,
3. $R \models P$.

Initially, $R = I$ and the algorithm checks if $\mathit{Pref}(j) \wedge \mathit{Suff}(k-j)$ is satisfiable. If it is the case, then there exists an initialized path (i.e. a path that starts from the initial states of S) that violates P. Otherwise, the algorithm computes the interpolant *itp* of the formulas $\mathit{Pref}(j)$ and $\mathit{Suff}(k-j)$. The interpolant *itp* is an over-approximation of all the states reachable in j steps and that cannot reach a bug in the next $k - j$ steps. If the formula $R \vee \mathit{itp}[V/V^0] \rightarrow R$ is valid, then the algorithm reached a fixed-point, proving P. Otherwise, the algorithm iterates relaxing the set if initial states $\mathit{Init} := \mathit{Init} \vee \mathit{itp}[V/V^0]$, thus analyzing an overapproximation of the original system (where $\mathit{itp}[V/V^0]$ denotes the formula obtained from *itp* replacing all the variables $v^0 \in V^0$ with the variable $v \in V$). Note that, if in the following iteration the algorithm finds that $\mathit{Pref}(j) \wedge \mathit{Suff}(k-j)$ is satisfiable, it cannot conclude that there exists a bug, since the set of initial states was approximated.

IC3

In this section we use the transition system definition without input variables and the invariant formula.

IC3 [Bra11] is an algorithm for checking invariant properties that in the last years proved to outperform the existing verification techniques for finite-state systems (e.g. see the results of the *Hardware Model Checking Competition* [BC10, BH11, BHSW12, BHSW13]). The algorithm is also known with the name *Property Directed Reachability* [EMB11]. We follow the presentation given in [CG12].

The goal of IC3 is either to find an inductive invariant $F(V)$ that proves that a symbolic transition system $S \models P$ or to determine the existence of a counterexample path. The inductive invariant is such that:

1. $Init(V) \models F(V)$,
2. $F(V) \wedge Trans(V, V') \models F(V')$,
3. $F(V) \models P(V)$.

IC3 keeps a sequence of sets of formulas $F_0, \ldots, F_k$, called *frames*, which over-approximate the set of the reachable states *up to* a fixed length (i.e. F_i over-approximates the set of states reachable by S in at most i steps). The algorithm keeps the following invariant conditions:

1. $F_0 = I$,
2. for all $i, 0 \leq i < k$, $F_i \rightarrow F_{i+1}$,

3. for all $i, 0 \leq i < k$, $F_i \wedge Trans \rightarrow F'_{i+1}$,
4. for all $i < k$, $F_i \models P$.

Frames are represented in CNF, and thus each frame is a set of clauses. This allows to rewrite conditions 2 using subset inclusion (i.e. for all $i, 0 < i < k$, $F_{i+1} \subseteq F_i$).

From an high level point of view, in each iteration IC3 performs two different phases, the *blocking phase* and the *propagation phase.* The goal of the *blocking phase* is to ensure that the last frame in the sequence, F_k, satisfies P (i.e. $F_k \models P$). The procedure is recursive and it either tries to prove that $F_k \models P$ or to prove the existence of a counterexample trace from the frame F_o to the frame F_k. Thus, in this phase the algorithm may conclude that $F_k \not\models P$. Then, in the *propagation phase* IC3 checks if a clause c in a frame F_i can be also added to the subsequent frame F_{i+1}. Intuitively, this can be done if any transition from one of the states represented by the clause c reaches only states in the same set. During this phase IC3 may discover that $F_{k-1} = F_k$, thus proving that $S \models P$. If it is not the case, IC3 adds a new frame to the sequence and iterates both phases.

IC3 relies on the concept of *relative induction* [Bra11].

Definition 6 (Inductive relative) *Given a transition system S and a formula $\phi(V)$, the formula $\psi(V)$ is* inductive relative *to ϕ if:*

$$\models Init \rightarrow \psi \tag{2.9}$$

$$\models Trans \wedge \phi \rightarrow \psi' \tag{2.10}$$

Thus, the basic step of IC3 consists of checking the relative inductiveness of a clause c with respect to a frame F_i:

$$\models F_i \wedge Trans \rightarrow c' \tag{2.11}$$

The algorithm is shown in Figure 2.1. Initially (line 1) the trace contains only one frame with the initial states. Then, the main IC3 loop (line 3) alternates the blocking (line 4) ad the propagation (line 7) phases. During the blocking phase IC3 checks if the current frame (i.e. the last frame added to the sequence, F_k satisfies P. If it is not the case, there exists a cube c that must be "blocked" by the previous frames. We denote *proof obligation* the pair (c, i), where c is a cube and i is an integer. In order to prove P, the algorithm must "discharge" the proof obligation, proving as an intermediate result that the cube c is not reachable in i steps. At line 5 IC3 calls the function *recBlock*. The function, also shown in Figure 2.1, recursively tries to discharge the proof obligation $(c, k-1)$. Note that, if $k = 0$ then the function actually found a counterexample. Otherwise, *recBlock* checks if $\neg c$ is relative inductive to F_{k-1}:

$$\models F_i \wedge Trans \wedge \neg c \wedge c' \tag{2.12}$$

This check tries to strengthen the induction adding $\neg c$. $\neg c$ holds in F_{i-1}, otherwise the algorithm would have found that $c \models F_{i-1} \wedge P$ in the $k-1$ step. If the check is satisfiable, then c is not inductive relative to F_{i-1}, and thus there exists a predecessor of c, say s, in F_{i-1} that can reach c in one step. The algorithm generates a new proof obligation $(s, k-1)$ that must be discharged (recursively). If c is inductive relative to F_{i-1} then the clause $\neg c$ could be *added* to F_k, thus discharging $(c, k-1)$. However, one of the strengths of IC3 is that it *generalizes* the blocking clause in order to rule out sets of states (otherwise, the approach would perform an explicit state exploration). We will briefly discuss about the the generalization step later.

Instead, if no bug where found in the frame F_k, IC3 extends the visited traces, adding F_{k+1} (line 7). At this point IC3 tries to propagate the clauses in a frame to the subsequent one. A clause c is propagated from a frame F_i to a frame F_{i+1} if c is inductive relative to F_i. During the propagation IC3 checks if two subsequent frames F_i and F_{i+1} have the same set of clauses. In this case F_i is an inductive invariant that proves $S \models P$.

The IC3 strength lies in the *generalization* step that is performed when blocking a proof obligation. Recall the relative inductive check of Equation 2.12. The generalization consists of "dropping" literals [BM07b] from the clause $\neg c$, keeping the relative inductive check for the new clause unsatisfiable. Intuitively, this has the effect of enlarging the set of states represented by the clause, and thus the new clause will rule out a larger set of states than $\neg c$. Bradley presented an algorithm [BM07b] that tries to "drop" the literals in a systematic way (the intuition is that one has to search in the lattice formed by all the possible sub-clauses of $\neg c$). This process can always use the unsatisfiable cores obtained from the SAT solver. Moreover, in [EMB11] the authors propose another way of "generalizing" the formulas manipulated by the algorithm. In particular, every time the algorithm get a counterexample to induction this may be "enlarged". The intuition is that a complete assignment contains *all* the state variables of S, and thus it represents a single state. Techniques such as three-valued ternary simulation and don't care detection may remove some variables from the complete assignment, thus generalizing the result.

Another interesting observation regards the implementation of the function *recBlock*. While the formulation of *recBlock* given in Figure 2.1 is recursive, the real implementations of the functions are iterative and based on a priority queue data structure, which stores the proofs obligations. The algorithm extracts the proof obligation from the queue, taking first the proof obligation with the higher index. With the priority queue, a proof obligation (c, i) that has to be blocked in frame i, with $0 < i \leq k$, it is also copied for all the indexes $i < j \leq k$ (i.e. $(c, i+1), \ldots, (c, k)$. The intuition is that the bad state c *must* be blocked also "in the future" by all the subsequent frames. This mechanism allows IC3 to find counterexample traces that are longer than the length of the frame sequence k.

IC3 can be applied also to first-order transition systems, just replacing the underlying SAT solver with an SMT solver. However, the resulting algorithm may be extremely inefficient and ineffective. The main reason lays in the missing generalization for the *theory* variables, which are not considered by the generalization. There have been a couple of implementation of IC3 to deal with systems represented in $\mathcal{T}(\mathbb{Q})$ [CG12, HB12]. The approach in [HB12] proposes an implementation of the generalization to induction based on interpolants. Suppose to have the (simplified) relative inductive query $F_i \wedge T \wedge \neg c'$. Then, if the query is unsatisfiable, the generalization computes the interpolant itp between the formulas $F_i \wedge T$ and $\neg c'$. By the property of interpolation, $\models (F_i \wedge T) \rightarrow itp$ and $\not\models \neg c \wedge itp$. Thus, itp clearly would block c in F_{i+1}. However, to fit the IC3 framework, itp must be a clause. This is solved in [HB12] computing a specific interpolant for $\mathcal{T}(\mathbb{Q})$, which is a clause by construction. A discussion on the computation of the interpolant is out of scope, we refer the interested reader to [HB12].

Instead, the approach in [CG12] focuses on the generalization of the counterexample to induction. The main issue with the counterexample to induction in the SMT case is that a cube s is of the form $s := s_1 \wedge \ldots \wedge s_{|V|}$, where $|V|$ is the cardinality of the set of state variables and each s_i is a predicate that assigns a value to a variable (e.g. $x = 1$). It is unlikely that the search converges since the number of possible assignments is infinite. Thus, instead of taking one single counterexample to induction, the proposed technique considers a set of counterexamples. Consider again the query $F_i \wedge T \wedge c'$. In the case it is satisfiable, one could compute the exact preimage of the set of states c' (i.e. $Pre(c') := \exists V'.(F_i \wedge T \wedge c')$)

and then take a cube in $Pre(c')$. A viable solution consists of performing an approximate quantifier elimination, APPROX-PREIMAGE, which returns just one cube $c \to Pre(c')$ (i.e. an under-approximation of the pre-image). Note that both solutions are orthogonal, and can be combined.

Specific implementation of IC3 have been proposed for the *Theory of Bit-vectors (UF_BV)* [WK13] and also for specific kinds of systems (e.g. timed-automata [KJN12b], well-structured transition systems [KMNP13]). A recent approach [CGMT14a] extends IC3 with implicit predicate abstraction [Ton09]. The approach does not require a specific generalization techniques for theories, since all the IC3 "machinery" is performed on the abstract state space. The only requirement is to have an effective refinement technique for the used theories (e.g. interpolation based [HJMM04]).

2.4 Hybrid Systems

2.4.1 Hybrid Automata

Hybrid Automata [Hen96] are a well known framework used to model hybrid systems. An hybrid automaton models both the discrete and the continuous behaviors of a system. The discrete state of the system is represented with a set of vertices of a graph, called *locations*, while real-valued functions, called *continuous variables*, model the continuous state. The discrete transitions are modeled using edges between two vertices, that are labelled with an expression that expresses when the transition is enabled and how the continuous variables change when the transition is executed. The continuous evolution, called *flow condition*, is described through differential equations over the continuous variables and their derivatives (with respect to time). Each discrete location has its own flow condition. Moreover, the continuous evolution is affected by the *invariant condition*, which expresses a relation over the continuous variables that must hold in each discrete location.

A state of the system is given by a location and an assignment to the continuous variables. The intuitive semantic of an hybrid automaton is to alternate discrete transitions, where the discrete location changes, to continuous transitions, where the time elapses and thus the value of the continuous variables change according to the flow and invariant conditions.

In the rest of the thesis, we denote with $\dot{f}$ the first derivative of a real function f and with $\dot{X}$ the set that contains the first derivatives of all the real functions in the set X (e.g. $\dot{X} = \{\dot{x} \mid x \in X\}$). As we did for symbolic transition system, we will denote with x' the value of the variable x in the next discrete execution step of the system. We extend this notation to a set of variables (e.g. $X' = \{x' \mid x \in X\}$).

We will use a definition of hybrid automaton where the discrete locations are not given explicitly, but are determined by the assignments to a set of discrete variables, and thus also the discrete transitions, the initial, the invariant and the flow conditions are expressed as first-order logic formulas.

Definition 7 (Hybrid Automaton [Hen96]) *A* Hybrid Automaton *(HA) is a tuple $\langle V, X, \varepsilon, Init, Invar, Trans, Flow\rangle$ where:*

- *V is a set of discrete variables,*
- *X is a set of continuous variables,*
- *ε is a variable with domain A,*
- *$Init$ is a $\Sigma_{\mathbb{R}}$-formula over $V \cup X$ (the initial condition),*
- *$Invar$ is a $\Sigma_{\mathbb{R}}$-formula over $V \cup X$ (the initial condition),*
- *$Trans$ is a $\Sigma_{\mathbb{R}}$-formula over $V \cup X \cup A \cup V' \cup X'$ (the transition relation),*
- *$Flow$ is a $\Sigma_{\mathbb{R}}$-formula over $V \cup X \cup \dot{X}$ (the flow condition).*

A *state* of a hybrid automaton H is an assignment s to the variables $V \cup X$. We refer to the set of values A, the domain of the variable ε, as *labels* of the hybrid automaton.

Definition 8 (Path) *A sequence* $s_0 \xrightarrow{\delta_1} s_1 \xrightarrow{\delta_2} \dots \xrightarrow{\delta_k} s_k$ *is a* path *of the hybrid automaton* H *if:*

- $s_0 \models Init$ *and for* $0 < i \leq k$, s_i *is a state of* H *and*
- *for* $1 \leq i \leq k$, $\delta_j \in \mathbb{R} \cup A$ *and* $s_{i-1} \xrightarrow{\delta_i} s_i$ *we have that either:*
  - *Discrete transition:* $\delta_i \in A$ *and* $\langle s_{i-1}, \delta_i, s_i \rangle \models Trans$, $s_{i-i} \models Invar$, $s_i \models Invar$.
  - *Continuous transition:* $\delta_i \in \mathbb{R}, \delta_i > 0$*:*
    - $s_{{i-1}_{|V}} = s_{i_{|V}}$,
    - *there exists a continuous differentiable function* $f : [0, \delta_i] \rightarrow \mathbb{R}^{|X|}$ *such that:*
      - $f(0) = s_{{i-1}_{|X}}$ *and* $f(\delta_i) = s_{i_{|X}}$,
      - $s_{i-1} \models Invar$, $s_i \models Invar$,
      - $\forall \epsilon \in [0, \delta_i]$, $\langle s_{{i-1}_{|V}}, f(\epsilon), \dot{f}(\epsilon) \rangle \models Flow$,
      - $\forall \epsilon \in [0, \delta_i], \langle s_{{i-1}_{|V}}, f(\epsilon) \rangle \models Invar$.

We say that a state s_k of the hybrid automaton H is *reachable* if there exists a run π of H such that $\pi = s_0 \xrightarrow{\delta_1} s_1 \xrightarrow{\delta_2} \dots \xrightarrow{\delta_k} s_k$.

While we use a symbolic definition of the hybrid automaton, we will denote with Q the set of all the assignments to the variables V. Formally, $Q = \prod_{v \in V} |Dom(v)|$, where $|Dom(v)|$ denotes the cardinality of the domain of the variable v. We extend the notation used to refer to the "primed" variables to assignments. If v is an assignment to the variables V, we denote with v' the same assignment to the variables in V' (e.g. if $v = \langle x = 1, y = 2 \rangle$, then $v' = \langle x' = 1, y' = 2 \rangle$). Q corresponds to the set of explicit locations of the hybrid automaton, as used by the standard definition [Hen96].

2.4.2 Classes of Hybrid Systems

We clarify the nomenclature used in this thesis when referring to the different kinds of hybrid systems that we consider:

- Hybrid systems with linear (or non-linear) constraints (e.g., [ACHH92, HHWT98]): linear and non-linear hybrid automata, where the flow condition is given by symbolic constraints over the derivatives of continuous variables.

- Hybrid systems with linear (or non-linear) ODE (see, e.g., [ADMB00, LPY01, Tiw08, FGD+11a]): following the literature on control theory, linear and non-linear hybrid systems where the flow condition is defined by a system of linear or non-linear Ordinary Differential Equations (ODE).

- Hybrid systems with polynomial (or non-linear) dynamics (see, e.g., [Frä01, CCF+07a, PC07]): hybrid systems such that the continuous evolution is described with a function of time, thus without using derivatives.

Our definition of hybrid systems is general enough to cover the first two cases. However, our results also applies to the third case, as such dynamics provide the explicit solution for a system of ODEs, as we require.

We formally define two main classes of systems, *Linear Hybrid Automata [Hen96]* and *Linear Hybrid Systems*. These two classes of systems have been widely used in the literature and are the target of several techniques presented in this thesis.

Definition 9 (Linear hybrid automata) *A* Linear Hybrid Automaton *(LHA) is an hybrid automaton such that:*

- *$Flow$ is a $\Sigma_{\mathbb{Q}}$-formula over $V \cup \dot{X}$,*
- *$Init$ is a $\Sigma_{\mathbb{Q}}$-formula over the variables $V \cup X$,*
- *$Invar$ is a $\Sigma_{\mathbb{Q}}$-formulas over the variables $V \cup X$ and for each location $q \in Q$, $Invar(q)$ is convex,*
- *$Trans$ is a $\Sigma_{\mathbb{Q}}$-formula over $V \cup X \cup A \cup V' \cup X'$.*

In some cases, we will specifically consider *linear dynamics*, where the flow condition $Flow$ for a specific location $q \in Q$ is a system of linear *Ordinary Differential Equations* (ODEs).

We denote column vectors with the the notation $\vec{x}$. We say that $\vec{x}$ is the vector of the set of variables X if $\vec{x}$ is the vector formed by all the variables $x \in X$, assuming that the order of the variables in the vector is chosen arbitrarily (in this thesis we assume a lexicographic order on the variables name). We will denote with $\vec{x}^T$ the transpose of the vector $\vec{x}$ (i.e. the row vector). Given a $m \times n$ matrix M, two positive integers i, j, we denote with $M_{i,j}$ the element in the $i-th$ row and $j-th$ column of M. In the case of column or row vector, we drop the constant index 1 from the notation (e.g. given $\vec{c}^T = [c_1, c_2, c_3]$, we write $\vec{c}_2 = c_2$ for $\vec{c}_{1,2}$).

We define a linear *Ordinary Differential Equations* as follows:

$$\dot{\vec{x}} = A\vec{x} + b$$

where $A \in \mathbb{R}^{n \times n}$ and $b : \mathbb{R} \to \mathbb{R}^n$, n is the cardinality of the set X (i.e. $|X| = n$), $\vec{x}$ is the vector of X and $\dot{\vec{x}}$ is the vector of X. Here, note that b is a function of time and thus may represent inputs that change over time (i.e. *non-autonomous system*). As remarked in [LPY01], this definition is sufficient to represent inputs of the form Bu where $B \in \mathbb{R}^{n \times m}$

and $u : \mathbb{R}^m \rightarrow \mathbb{R}^n$. We will write $Flow(q) = A\vec{x} + b$ to refer to the dynamic of the location $q \in Q$. We deal with this kind of dynamics in Section 3.4.

Moreover, we define *Linear Hybrid System* a hybrid automaton where the flow condition in each location is linear and where all the constraints of the automaton are expressed in $\mathcal{T}(\mathbb{Q})$.

Definition 10 (Linear Hybrid System) *With the term* Linear Hybrid System *(LHS) we refer to a an hybrid automaton such that, for all the locations $q \in Q$:*

- *The dynamic in each location $q \in Q$ is a system of linear ODEs,*
- *$Init$ is a $\Sigma_{\mathbb{Q}}$-formula over the variables $V \cup X$,*
- *$Invar$ is a $\Sigma_{\mathbb{Q}}$-formulas over the variables $V \cup X$ and for each location $q \in Q$, $Invar(q)$ is convex,*
- *$Trans$ is a $\Sigma_{\mathbb{Q}}$-formula over $V \cup X \cup A \cup V' \cup X'$.*

In Chapter 5 we deal with *autonomous* linear hybrid systems, where the inputs b do not depend on time. Due to the restriction, the dynamic $Flow(q)$ can be written as:

$$Flow(q) := A\vec{x} + \vec{b}$$

where $A \in \mathbb{R}^{n \times n}$, $\vec{x}$ is the vector of X, $\dot{\vec{x}}$ is the vector if $\dot{X}$, $\vec{b} : \mathbb{R}^n$, n is the dimension of the dynamical system (i.e. the cardinality of the set X, $|X| = n$) and $\vec{X}$ is the vector of all the continuous variables.

2.4.3 Hybrid Automata Network

To represent concurrent systems, we consider the asynchronous composition of hybrid automata.

Definition 11 (Parallel Composition of Hybrid Automata) *Given two hybrid automata* $H_1 = \langle V_1, X_1, \varepsilon_1, Init_1, Invar_1, Trans_1, Flow_1 \rangle$ *and* $H_2 = \langle V_2, X_2, \varepsilon_2, Init_2, Invar_2, Trans_2, Flow_2 \rangle$*, where* $V_1 \cap V_2 = \emptyset$ *and* $X_1 \cap X_2 = \emptyset$*, their parallel composition* $H_1 || H_2$ *is the hybrid automaton* $H = \langle V, X, \varepsilon, Init, Invar, Trans, Flow \rangle$ *where:*

- $V := V_1 \cup V_2$,
- $X := X_1 \cup X_2$,
- ε *is a variable such that its domain* $A = A_1 \cup A_2$,
- $Init := Init_1 \wedge Init_2$,
- $Invar := Invar_1 \wedge Invar_2$,
- $Trans := \bigvee_{e \in A_1 \setminus A_2} (\varepsilon = e \wedge Trans_1 \wedge V_2' = V_2 \wedge X_2' = X_2) \vee$
  $\bigvee_{e \in A_2 \setminus A_1} (\varepsilon = e \wedge Trans_2 \wedge V_1' = V_1 \wedge X_1' = X_1) \vee$
  $\bigvee_{e \in A_2 \cap A_1} (\varepsilon = e \wedge Trans_1 \wedge Trans_2)$
- $Flow = Flow_1 \wedge Flow_2$.

A *network* $\mathcal{N}$ of HAs is the parallel composition of two or more HAs: $\mathcal{N} = H_1 || \ldots || H_n$, where $H_1, \ldots, H_n$ are hybrid automata. In the following, we consider a network $\mathcal{N} = \mathcal{N} = H_1 || \ldots || H_n$ of HAs with $H_i = \langle V_i, X_i, \varepsilon_i, Init_i, Invar_i, Trans_i, Flow_i \rangle$ such that for all $1 \leq i < j \leq n$ $X_i \cap X_j = \emptyset$ and $V_i \cap V_j = \emptyset$ (i.e. both the set of continuous and discrete variables of the hybrid automata are disjoint). Other definitions of hybrid automata consider shared variables, for example *I/O Hybrid Automata* [LSV03].

```
bool IC3-prove($Init$, $Trans$, $P$):
1.  trace = [$Init$] # first elem of trace is init formula
2.  trace.push() # add a new frame to the trace
3.  while True:
        # blocking phase
4.      while there exists a cube $c$ s.t. $c \models trace.last() \wedge \neg P$:
            # recursively block the pair $(c, trace.size() - 1)$
5.          if not recBlock(c, trace.size()-1):
                # a pair $(s_0, 0)$ is generated
6.              return False # counterexample found

        # propagation phase
7.      trace.push()
8.      for $i = 1$ to trace.size() $- 1$:
9.          for each clause $c \in$ trace[i]:
10.             if trace[i] $\wedge c \wedge Trans \wedge \neg c'$ is unsatisfiable:
11.                 add $c$ to trace[i+1]
12.         if trace[i] == trace[i+1]:
13.             return True # property proved

# simplified recursive description, in practice based on priority queue [Bra11, EMB11]
bool recBlock($s, i$):
1.  if $i = 0$: return False # reached initial states
2.  while there exists a cube $c$ such that $\models F_i \wedge Trans \wedge c \wedge s'$:
3.      if not recBlock($c, i - 1$): return False
4.  $g$ = generalize($\neg s$, i) # standard IC3 generalization [Bra11, EMB11]
5.  add $g$ to trace[i]
6.  return True
```

Figure 2.1: High-level description of IC3 (following [EMB11]).

Part II

Encoding Techniques

Chapter 3

Hybrid Automata Encoding

Note. The material presented in this chapter has already been presented in [CMT12, CMT13b].

In this chapter we present the encoding of a hybrid automaton H in a symbolic transition system S. The encoding preserves safety properties [1]: a safety property P holds in the hybrid automaton H ($H \models P$) if and only if P holds in the symbolic transition system S. Thus, after the encoding we can apply all the algorithms presented in Section 2.3 to prove P.

We will first introduce a general encoding in the theory of reals extended with transcendental function and universal quantifiers. While this definition is general, it cannot be used in a practical setting. The definition uses a very expressive theory, $\mathcal{T}(\overline{\mathbb{R}})$, that is not currently handled by the existing SMT solvers. We will see that, for many systems of practical interest, we do not need this theory and we rely on $\mathcal{T}(\mathbb{R})$ or $\mathcal{T}(\mathbb{Q})$. From a practical point of view, the experimental results presented in Chapter 10 shows benchmarks that use $\mathcal{T}(\mathbb{Q})$ and $\mathcal{T}(\mathbb{R})$.

The encoding assumes the existence of an explicit solution for the *Flow* condition of the hybrid automaton that can be expressed in $\mathcal{T}(\overline{\mathbb{R}})$. We may find an explicit solution for several classes of systems. In particular, this is true for *Linear Hybrid Automata*, for *Polynomial Hybrid Systems* and

[1]The presented encoding preserves also LTL_X, the fragment of LTL without the next operator

for some sub-classes of *Linear Hybrid Systems*. Moreover, some non-linear hybrid systems may have an explicit solution in $\mathcal{T}(\overline{\mathbb{R}})$. The continuous evolution is encoded as a transition of the symbolic transition system, where the real time elapses and the continuous variables change their values according to $Trans$. Note that the "continuous" transition of the encoding is point-wise: it expresses the fact that the system changes its state from $s(t)$ to $s(t')$.

The encoding also forces that the invariant condition $Invar$ holds in all the time points in the interval $[t, t']$, i.e. $\forall \epsilon \in [t, t'], Invar(\epsilon)$. This formula has a universal quantifier to encode that the continuous (timed) transition is always enclosed in the invariant condition of the hybrid automata. The handling of these quantifiers is an open problem [ÁBKS05], since the quantifier elimination problem for $\mathcal{T}(\overline{\mathbb{R}})$ it not decidable, while for $\mathcal{T}(\mathbb{R})$ the existing quantifier-elimination procedures such as CAD [Col75] do not scale.

We show a technique that allow us to obtain a *quantifier-free* encoding for several interesting classes of hybrid systems. The main idea consists of forcing the invariant to hold before and after the continuous transition (i.e. $Invar(t)$ and $Invar(t')$), and the sign of the derivative of the invariant to be constant during a continuous transition. This reduction yields a universally quantified formula over the derivative of the invariant, and not over the invariant itself. In several interesting cases, the reduction may be applied recursively until the quantifiers are applied to a linear function. At this point, the universal quantifiers can be safely removed by convexity [HKPV98], obtaining a quantifier free formula.

Finally, we show that we can encode different classes of systems with polynomial and linear dynamics using $\mathcal{T}(\mathbb{R})$. While the SMT encoding of systems for simpler dynamic like linear hybrid automata is straightforward, and well know in the literature, the encoding for the other classes of systems exploits our reduction that removes the quantifiers. Thus, the encoding extends the applicability of the SMT-based verification methods to these classes of systems.

3.1 Encoding of a single automaton

We show the encoding of a single hybrid automaton in a symbolic transition system.

In this chapter we focus on single hybrid automaton and thus, to simplify the notation, we disregard the labels on the discrete transitions of the automaton (i.e. the automaton is defined as a tuple $\langle V_H, X_H, Init_H, Invar_H, Trans_H, Flow_H\rangle$, where $Trans$ is a $\Sigma_{\mathbb{R}}$-formula over $V \cup X \cup V' \cup X'$).

In our hybrid automata definition the $Flow$ condition is a general predicate over V, X and $\dot{X}$. The encoding considers hybrid automata where the continuous dynamics $Flow$ is described by a system of Ordinary Differential Equations [2] (ODEs) in the form $\dot{X} = F(X)$ (i.e., for all $x \in X$, $\dot{x} = F_x(X)$). Then, we assume that the system of ODEs admits a primitive solution $f(V, t)$, which is uniquely determined by the state at the beginning of the timed transition.

Let $\langle V_H, X_H, Init_H, Invar_H, Trans_H, Flow_H\rangle$ be an hybrid automaton. Then, the encoding of H is the first-order transition system $S_H = \langle V_S, \mathcal{W}_S, Init_S, Inv_S, Trans_S\rangle$ defined as:

- $V_S := V \cup X \cup \{t\}$,
  (t is a real variable used to track the amount of time elapsed)
- $\mathcal{W}_S := \{\}$,
- $Init_S := t = 0 \wedge Init_H$,

[2]In the case of *Linear hybrid automata* we also allow more general predicates in the flow condition, like $\dot{x} + \dot{y} <= 5$ (See Example 2).

- $Inv_S := Invar_H$
  Note that Inv_S does not ensure that $Invar_H$ holds during a continuous transition. This is encoded by the universal quantifier of $Trans_S$.

- $Trans_S :=$ Untimed $\vee$ Timed where
  - Untimed $:= t' = t \wedge Trans_H(V, X, V', X')$.
  - Timed $:= t' > t \wedge V' = V \wedge X' = f(V, t') \wedge$
    $\forall \epsilon \in [t, t'], Invar_H(V, f(V, \epsilon))$

From now on, we will call S_H the *encoding* of H.

Theorem 1 *Given a HS H, the symbolic transition system $S_H = \langle V_S, \mathcal{W}_S, Init_S, Inv_S, Trans_S \rangle$ is such that there exists a one-to-one mapping between the paths of H and the paths of S_H.*

Proof. ($\Rightarrow$) Let $\pi_H = h_0 \xrightarrow{\delta_1} h_1 \xrightarrow{\delta_2} \ldots \xrightarrow{\delta_k} h_k$ be a path of the hybrid automaton H. Then, the sequence of states $\pi_S = s_0; a_1; s_1; \ldots; a_k; s_k$ such that:

- for all $0 \leq i \leq k$, $\forall v \in V, s_i(v) = h_i(v)$,
- for all $0 \leq i \leq k$, $\forall x \in X, s_i(x) = h_i(x)$,
- $s_0(t) = 0$,
- for all $0 < i \leq k, s_i(t) \begin{cases} s_{i-1}(t) & \text{if } \delta_i \notin \mathbb{R}, \\ s_{i-1}(t) + \delta_i & \text{otherwise} \end{cases}$

is a path of S_H.

Clearly, $s_0 \models Init_S$ and for all $0 \leq i \leq k$, $s_i \models Inv_S$ (since $h_0 \models Init_H$ and $s_0(t) = 0$ and for all $0 \leq i \leq k$ $h_i \models Invar_H$). Then, if $h_{i-1} \xrightarrow{\delta_i} h_i$ is a discrete transition we have that $\langle s_{i-1}, s_i \rangle \models$ Untimed (note that $s_{i-1}(t) = s_i(t)$). Otherwise, if $h_{i-1} \xrightarrow{\delta_i} h_i$ is a continuous transition

$\langle s_{i-1}, s_i \rangle \models$ TIMED. Since $h_{i-1} \xrightarrow{\delta_i} h_i$ with a continuous transition, we have that:

- $\delta_i > 0$, and thus $s_{i-1}(t) > s_i(t)$, and thus $t' \wedge t$ holds,
- $h_{i-1_{|V}} = h_{i_{|V}}$, and thus $s_{i-1_{|V}} = s_{i_{|V}}$, and thus $V'_S = V_S$ holds,
- $h_{i_{|X}} = f(h_{i-1_{|discvars}}, \delta)$ and thus $\langle s_{i-1}, s_i \rangle \models X' = f(V, t')$.
- $\forall \epsilon \in [0, \delta], f(h_{i-1_{|discvars}}, \epsilon) \models Invar_H$, and thus $\langle s_{i-1}, s_i \rangle \models \forall \epsilon \in [t, t'], Invar_H(V, f(V, \epsilon))$ holds.

($\Leftarrow$) Let $\pi_S = s_0; a_1; s_1; \ldots ; a_k; s_k$ be a path of S_H. Then, the sequence of states $\pi_H = h_0 \xrightarrow{\delta_1} h_1 \xrightarrow{\delta_2} \ldots \xrightarrow{\delta_k} h_k$ such that:

- for all $0 \leq i \leq k$, $\forall v \in V, s_i(v) = h_i(v)$,
- for all $0 \leq i \leq k$, $\forall x \in X, s_i(x) = h_i(x)$,
- for all $0 < i \leq k, \delta_i = s_i(t) - s_{i-1}(t)$

is a path of H. The proof is similar to the one in the previous case (i.e. we can easily prove that $h_i \models Init_H$, $h_i \models Invar_H$ and that $h_{i-1} \xrightarrow{\delta_1} h_i$ is a transition of H, either discrete or continuous). $\diamond$

Example 2 (Encoding of Linear Hybrid Automaton) *In the case H is a Linear Hybrid Automaton, the encoding [ABCS05, dMRS02, ÁBKS05] is the transition system:*

- $V_S := V \cup X \cup \{t\}$,
- $\mathcal{W}_S := \{\}$,
- $Init_S := t = 0 \wedge Init_H$,
- $Inv_S := Invar_H$,

- $Trans_S :=$ UNTIMED $\vee$ TIMED *where*
  - UNTIMED $:= t' = t \wedge Trans_H(V, X, V', X')$.
  - TIMED $:= t' > t \wedge V' = V \wedge \gamma(V, X, t, X', t')$

where γ *is a* $\Sigma_{\mathbb{Q}}$*-formula over variables* $V \cup X \cup X' \cup \{t, t'\}$*, obtained copying the Flow expression replacing each* $\dot{x}$ *occurrence by* $\frac{x-x}{t'-t}$ *and removing the denominator* $t' - t$ *(note that* $t' > t$*). Formally, since Flow contains predicate of the form* $\sum_{x \in X} a_x \cdot \dot{x} \bowtie b$*, for some* $a_x \in \mathbb{R}, b \in \mathbb{R}, \bowtie \in \{<, \leq, >, \geq, =, \neq\}$ *we have that* $\gamma = \sum_{x \in X} a_x \cdot (x' - x) \bowtie (t' - t) \cdot b$*. Note that the universal quantification that encodes the invariant condition of the automaton can be removed since Flow is linear and the invariant Inv is convex [HKPV98].*

Remark 2 (Alternative encoding without t) *In several cases we are not interested to keep track of the amount of time elapsed during the automaton run. In these cases, we avoid to add* t *to* V_S *and we use an additional input variable* δ *of real type* ($\mathcal{W}_S := \{\delta\}$)*.* δ *represents only the amount of time elapsed during the last continuous transition. Thus, the encoding forces* $\delta > 0$ *during* TIMED*. This different version of the encoding may be more effective when using induction methods (e.g. k-induction).*

3.2 Quantifier-free encoding for non-linear hybrid automata

In this section we show the reduction of the encoding with universal quantifiers to a quantifier-free encoding.

The reduction assumes that the invariant condition $Invar$ does not contain disjunctions. However, we do not restrict the form of $Invar$ but we show that the universal quantifier can be distributed over disjunctions.

While in general this is not possible for arbitrary first-order formulas, it preserves the hybrid automata semantic. The handling of disjunctive invariants requires an universal quantification over *open intervals* (e.g. $\forall \epsilon \in (t, t')$). This does not change the semantic of the system, since the invariant condition also forced to hold on the points t and t' (i.e. $Invar(t)$ and $Invar(t')$).

Thus, in the following section we consider an encoding where the continuous transition is encoded as:

$$\textsc{Timed} := t' > t \wedge V' = V \wedge X' = f(V, t') \wedge \forall \epsilon \in (t, t'), Invar(V, f(V, \epsilon))$$

The reduction is formalized by *hybrid traces* [dAM95, CRT09] that allow us to handle an arbitrary combinations of intervals, as opposed to the definition of hybrid automata paths that only use closed intervals ($[t, t']$) or left-closed and right-open intervals ($[t, t')$).

We will first introduce hybrid traces and then we show how we can handle disjunctive invariants. Then we present the reduction that enables us to remove the universal quantifier and several examples of its application.

3.2.1 Hybrid traces

Let I be an interval of $\mathbb{R}$ or $\mathbb{N}$; we denote with $le(I)$ and $ue(I)$ the lower and upper endpoints of I, respectively.

Hybrid traces [dAM95, CRT09] describe the evolution of variables in every point of time. Such evolution is allowed to have a countable number of discontinuous points corresponding to changes in the discrete part of the model.

Definition 12 (Hybrid Trace) *A* hybrid trace *over discrete variables V and continuous variables X is a sequence $\langle \overline{f}, \overline{I} \rangle := \langle f_0, I_0 \rangle, \langle f_1, I_1 \rangle, \ldots, \langle f_k, I_k \rangle$ such that, for all i, $0 \leq i \leq k$,*

- *the intervals are adjacent, i.e.* $ue(I_i) = le(I_{i+1})$*;*
- $le(I_0) = 0$ *and* I_k *is right closed;*
- $f_i : V \cup X \to \mathbb{R} \to \mathbb{R}$ *is a function such that, for all* $x \in X$*,* $f_i(x)$ *is differentiable, and for all* $v \in V$*,* $f_i(v)$ *is constant;*
- *if* I_i *is left open and* $le(I_i) = t$ *then, for all* $v \in V \cup X$*,* $f_i(v)(t) = f_{i-1}(v)(t)$*.*
- *Similarly, if* I_i *is right open and* $ue(I_i) = t$ *then, for all* $v \in V \cup X$*,* $f_i(v)(t) = f_{i+1}(v)(t)$*.*

We say that a trace is a sampling refinement of another one if it has been obtained by splitting an open interval into two parts by adding a sampling point in the middle [dAM95]. A *partitioning function* μ is a sequence $\mu_0, \mu_1, \mu_2, \ldots$ of non-empty, adjacent and disjoint intervals of $\mathbb{N}$ partitioning $\mathbb{N}$. Formally, $\bigcup_{i \in \mathbb{N}} \mu_i = \mathbb{N}$ and $ue(\mu_i) = le(\mu_{i+1}) - 1$.

Definition 13 (Sampling Refinement) *A hybrid trace* $\langle \overline{f}', \overline{I}' \rangle$ *is a* sampling refinement *of* $\langle \overline{f}, \overline{I} \rangle$ *(denoted with* $\langle \overline{f}', \overline{I}' \rangle \preceq \langle \overline{f}, \overline{I} \rangle$*) iff, there exists a partitioning* μ *such that for all* $i \in \mathbb{N}$*,* $I_i = \bigcup_{j \in \mu_i} I'_j$ *and, for all* $j \in \mu_j$*,* $f'_j = f_i$*.*

We extend the relation to set L_1 and L_2 of traces as follows: $L_1 \preceq L_2$ iff for every trace $w_2 \in L_2$ there exists $w_1 \in L_1$ such that $w_1 \preceq w_2$.

We assume that the evolution of predicates along time have the finite variability property. We say that a predicate $P(t)$ over a real variable t has *finite variability* [Rab98] iff for any bounded interval J there exists a finite sequence of real numbers $t_0 < \ldots < t_n$ such that $t_0 = le(J)$, $t_n = ue(J)$, and for all $i \in [1, n]$, either for all $\epsilon \in (t_{i-1}, t_i)$, $P(\epsilon)$ or for all $\epsilon \in (t_{i-1}, t_i)$, $\neg P(\epsilon)$. The last condition means that the predicate is constant in the interval (t_{i-1}, t_i). If P is in the form $g(t) \bowtie 0$ with g continuous and $\bowtie \in \{\geq, \leq, <, >\}$, in the points in which P changes value, $g(t) = 0$.

Definition 14 (Finite Variability) *A predicate $g \bowtie 0$ has finite variability iff for any bounded interval J there exists a finite sequence of real numbers $t_0 < \ldots < t_n$ such that $t_0 = le(J)$, $t_n = ue(J)$, and for all $i \in [1, n]$, either for all $\epsilon \in [t_{i-1}, t_i]$, $g(\epsilon) \geq 0$ or for all $\epsilon \in [t_{i-1}, t_i]$, $g(\epsilon) \leq 0$. We denote this condition with $Constant(P, t_{i-1}, t_i)$.*

Proposition 1 *Assuming that a predicate P has finite variability, for every hybrid trace σ, there exists a sampling refinement of σ for which P is constant in the open part of every interval.*

Given a hybrid trace $\langle f_0, I_0\rangle, \langle f_1, I_1\rangle, \ldots, \langle f_k, I_k\rangle$, we denote with $s_{f_i}(t)$ the state assigning to every variable $v \in V \cup X$ the value $f_i(v)(t)$ and with $\dot{s}_{f_i}(t)$ the assignment that maps every variable $v \in X$ with the value $\dot{f}_i(v)(t)$.

A hybrid trace $\langle f_0, I_0\rangle, \langle f_1, I_1\rangle, \ldots, \langle f_k, I_k\rangle$ is a *path* of the hybrid automaton $H = \langle V, X, Init, Invar, Trans, Flow\rangle$ iff:

- $s_{f_0}(0)$ satisfies $Init$;
- for every $0 \leq i \leq k$, for all $t \in I_i$, $s_{f_i}(t)$ satisfies $Invar$;
- for every $0 \leq i < k$, if I_i is right closed with $ue(I_i) = t$ and I_{i+1} is left closed with $le(I_{i+1}) = t'$, then $s_{f_i}(t), s'_{f_{i+1}}(t')$ satisfies $Trans$;
- for every $0 \leq i \leq k$, for all $t \in I_i$, $s_{f_i}(t), \dot{s}_{f_i}(t)$ satisfy $Flow$.

The *language* $L(S)$ is the set of models of S.

Proposition 2 *A sampling refinement of a path of an HS S is also a path of S.*

Intuitively, sampling refinement just splits an interval into sub-intervals and therefore does not change either the initial state or the discrete transitions. Thus, the conditions remain satisfied by the corresponding points.

Sampling refinement preserves reachability properties in the sense that if $L' \preceq L(S)$ then there exists a trace in L' reaching a condition ϕ iff there exists a trace in $L(S)$ reaching ϕ (similarly for LTL properties without next operators [AN95] and HRELTL properties [CRT09]).

3.2.2 Removing quantified disjunctions from the invariants

In this section we describe how to remove disjunctions from the invariants, obtaining an equivalent encoding where the quantified formulas contain only atomic predicates. The transformation relies on the encoding of hybrid systems with quantifiers over open intervals. Then, in the next Section we will show how to remove the quantifiers in both the open and the closed intervals case. Note that the existing encodings of hybrid automata into infinite-state transition systems ignore the issue and assume the convexity of the invariant condition.

We reduce the quantification over a disjunctive invariant into a disjunction of quantifications. While this is not correct in general, it is possible due to the particular position of the quantified sub-formula in the transition condition. After the reduction we guarantee that the quantified formula in $Trans_S$ is atomic, allowing us to remove the quantifiers.

Suppose we have a disjunctive invariant $\phi(\epsilon) \vee \psi(\epsilon)$. In our case we can distribute the universal quantifier in $\forall \epsilon \in (t,t'), \phi(\epsilon) \vee \psi(\epsilon)$ over the disjunction, obtaining the formula $\forall \epsilon \in (t,t'), \phi(\epsilon) \vee \forall \epsilon \in (t,t'), \psi(\epsilon)$. The following theorem proves the correctness of the transformation.

Theorem 2 *Let H be a hybrid automaton with $Invar = \phi(\epsilon) \vee \psi(\epsilon)$, where the predicates ϕ and ψ have finite variability, and S_H its encoding. Then, the encoding S'_H obtained from S_H replacing the subformula $\forall \epsilon \in (t,t'), \phi(\epsilon) \vee \psi(\epsilon)$ in $Trans_S$ with $\forall \epsilon \in (t,t'), \phi(\epsilon) \vee \forall \epsilon \in (t,t'), \psi(\epsilon)$ is sampling refinement to the original hybrid automaton.*

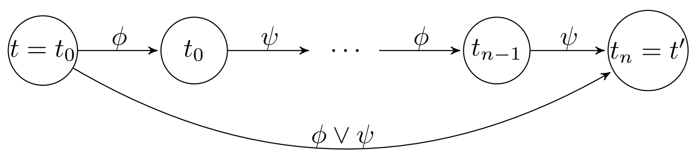


Figure 3.1: Effect on a path of the encoding without disjunctions.

Proof. ($\Leftarrow$) Clearly, $\forall \epsilon \in (t, t'), \phi(\epsilon) \vee \forall \epsilon \in (t, t'), \psi(\epsilon)$ implies $\forall \epsilon \in (t, t'), \phi(\epsilon) \vee \psi(\epsilon)$.

($\Rightarrow$) Consider a hybrid trace $\langle \overline{f}, \overline{I} \rangle := \langle f_0, I_0 \rangle, \langle f_1, I_1 \rangle, \ldots, \langle f_k, I_k \rangle$ which is a path of a H. Assuming that the predicates ϕ and ψ have finite variability, we can refine the hybrid trace into a new hybrid trace $\langle \overline{f'}, \overline{I'} \rangle' := \langle f'_0, I'_0 \rangle, \langle f'_1, I'_1 \rangle, \ldots, \langle f'_l, I'_l \rangle$ in which ϕ and ψ are constant in every interval. The new hybrid trace also satisfies S_H by Proposition 2 and thus the corresponding discrete trace $s_0; \ldots; s_l$ satisfies its encoding S'_H. At every i, if s_i satisfies $\forall \epsilon \in (t, t'), \phi(\epsilon) \vee \psi(\epsilon)$, then $f(s_i, \epsilon)$ satisfies $\phi \vee \psi$ for all $\epsilon \in (t, t') = (le(I'_i), ue(I'_i))$, and thus either ϕ or ψ (since ϕ and ψ are constant in the open part of I'_i). Therefore the discrete trace satisfies also the encoding with $\forall \epsilon \in (t, t'), \phi(\epsilon) \vee \forall \epsilon \in (t, t'), \psi(\epsilon)$. $\diamond$

The effect of the encoding on the paths of the transition system is shown in Figure 3.1. In practice, the encoding of the transition system without disjunctions splits the continuous transition every time the valuation of ϕ or ψ changes, instead of allowing a single continuous transition where $\phi \vee \psi$ holds.

3.2.3 Reduction to flow invariants

In this section we present a result that allow us to reduce the quantified formula of an invariant to a quantified formula over its derivatives. The application of the reduction results in a formula where the quantifier is applied to the derivatives of the invariant. Thus, the reduction could be applied recursively to obtain a quantified formula over the second order derivatives. In some cases, the quantifiers on the derivatives may be removed trivially, obtaining a quantifier-free formula (e.g. this happens when the derivative is a linear predicate).

The following theorems assume the finite variability of predicates of the derivatives. Many functions have this property, in particular polynomials and some simple transcendental functions.

Theorem 3 *If $g : \mathbb{R} \to \mathbb{R}$ is a differentiable function and $\dot{g} \bowtie 0$ ($\bowtie \in \{\geq, >, \leq, <\}$) has finite variability, then $\forall \epsilon \in [t, t'], g(\epsilon) \bowtie 0$ iff there exists a finite sequence of real numbers $t = t_0 < \ldots < t_n = t'$ such that $\bigwedge_{0 \leq i \leq n} g(t_i) \bowtie 0 \wedge \bigwedge_{0 < i \leq n} Constant(\dot{g} \geq 0, t_{i-1}, t_i)$.*

Proof. Let us assume that $\bowtie \in \{\geq, >\}$.

($\Rightarrow$) Since $\dot{g} \bowtie 0$ has finite variability, there exists a finite sequence of real numbers $t_0 = t < \ldots < t_n = t'$ such that $\bigwedge_{0<i\leq n} Constant(\dot{g} \geq 0, t_{i-1}, t_i)$ by definition. Moreover, since $\forall \epsilon \in [t, t'], g(\epsilon) \bowtie 0$, $g \bowtie 0$ holds also in the time points $t_0, \ldots, t_n$.

($\Leftarrow$) Assume by contradiction that there exists $t_b \in [t, t']$ such that $g(t_b) \bowtie 0$ is false. Since $\bigwedge_{0 \leq i \leq n} g(t_i) \bowtie 0$, there exists $i \in [1, n]$ such that $t_b \in (t_{i-1}, t_i)$. Since g is differentiable, by the mean value theorem, there exists a point $t'_b \in (t_{i-1}, t_b)$ such that $\dot{g}(t'_b) = \frac{g(t_b) - g(t_{i-1})}{(t_b - t_{i-1})}$ and therefore $\dot{g}(t'_b) < 0$. Similarly, there exists a point $t''_b \in (t_b, t_i)$ such that $\dot{g}(t''_b) = \frac{g(t_i) - g(t_b)}{(t_i - t_b)}$ and therefore $\dot{g}(t''_b) > 0$. Thus, $\dot{g}$ is not constant over (t_{i-1}, t_i) contradicting the hypothesis. We conclude that $\forall \epsilon \in [t, t'], g(\epsilon) \bowtie 0$.

The cases in which $\bowtie \in \{\leq, <\}$ can be proved similarly. $\diamond$

The intuition behind the theorem is simple. While we can easily encode that the invariant holds at some precise point (e.g. t, t') it is harder to impose the same condition along an interval without the use of quantifiers. However, we exploit the sign of $\dot{g}$ to infer the behavior of g in $[t, t']$ (i.e. if

$\dot{g} > 0$, resp. $\dot{g} < 0$, resp. $\dot{g} = 0$, then g increases, resp. decreases, resp. is constant). Since g is finite variable, we can divide the interval $[t, t']$ in a finite sequence of intervals where the sign of the derivative is constant. If the invariant does not hold in all the endpoints of the intervals, then there exists a point in $[t, t']$ where the invariant does not hold. Otherwise, by the fact that the sign of the derivative is constant we know that g just increases, decreases or is constant in the interval. Thus, the valuation of $g \bowtie 0$ does not change in the interval.

Example 3 *Consider the function $g(t) = t^3 - 3t$ in the interval $[-2, 2]$ (it is plotted in Figure 3.2) and the invariant $g(t) \geq 0$. The derivative $\dot{g}(t) < 0$ in $[-2, -1)$ and $(1, 2]$, $\dot{g}(t) > 0$ in $(-1, -1)$, $\dot{g}(t) = 0$ in $[-1, -1]$ and $[1, 1]$. Now consider the formula $\forall \epsilon \in [-2, 0], g(\epsilon) \geq 0$. We remove the quantifier considering the intervals where the derivative is constant: $g(-2) \geq 0 \wedge g(-1) \geq 0 \wedge g(0) \geq 0 \wedge Constant(\dot{g} \geq 0, -2, -1) Constant(\dot{g} \geq 0, -1, 0)$. Note that in practice we do not fix the intervals where the derivative is constant, but they will be found automatically by the SMT solver.*

When the predicate is an equality, the reduction is simpler.

Corollary 1 *If $g : \mathbb{R} \to \mathbb{R}$ is a differentiable function and $\dot{g} = 0$ has finite variability, then $\forall \epsilon \in [t, t'], g(\epsilon) = 0$ iff $g(t) = 0 \wedge g(t') = 0 \wedge \forall \epsilon \in [t, t'], \dot{g}(\epsilon) = 0$.*

The quantifier can be removed even if the interval of quantification is open.

Theorem 4 *If $g : \mathbb{R} \to \mathbb{R}$ is a differentiable function and $\dot{g} \bowtie 0$ ($\bowtie \in \{\geq, >\}$) has finite variability, then $\forall \epsilon \in (t, t'), g \bowtie 0$ iff there exists a finite set of real numbers $t = t_0 < \ldots < t_n = t'$ such that $g(t) \geq 0 \wedge g(t') \geq 0 \wedge \bigwedge_{0<i<n} g(t_i) \bowtie 0 \wedge \bigwedge_{0<i\leq n} Constant(\dot{g} \geq 0, t_{i-1}, t_i)$ if $\bowtie = \geq$, $g(t) \geq 0 \wedge g(t') \geq 0 \wedge \bigwedge_{0<i<n} g(t_i) \bowtie 0 \wedge \bigwedge_{0<i\leq n} Constant(\dot{g} \geq 0, t_{i-1}, t_i) \wedge (g(t) = 0 \to \dot{g}(t) > 0) \wedge (g(t') = 0 \to \dot{g}(t) < 0)$, if $\bowtie = >$.*

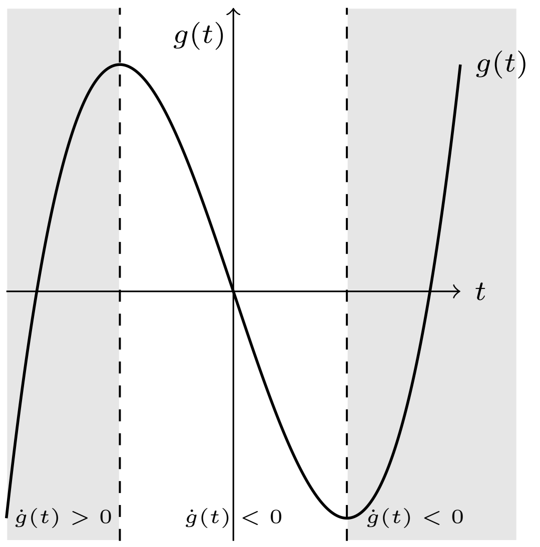


Figure 3.2: Plot of the function $t^3 - 3t$. The plot is gray if $\dot{g}(t) > 0$ and white when $\dot{g}(t) < 0$.

Proof. ($\Rightarrow$) Since $\dot{g} \bowtie 0$ has finite variability, there exists a finite set of real numbers $t = t_0 < \ldots < t_n = t'$ such that $\bigwedge_{0<i\leq n} Constant(\dot{g} \geq 0, t_{i-1}, t_i)$ by definition. Moreover, since $\forall \epsilon \in (t, t'), g \bowtie 0$, $g \bowtie 0$ holds also in the time points $t_1, \ldots, t_{n-1}$. $g(t) \geq 0$ and $g(t') \geq 0$ for the continuity of g. Finally, $(g(t) = 0 \rightarrow \dot{g}(t) > 0) \wedge (g(t') = 0 \rightarrow \dot{g}(t) < 0)$, if $\bowtie = >$.

($\Leftarrow$) Assume by contradiction that there exists $t_b \in (t, t')$ such that $g(t_b) \bowtie 0$ is false. Since $\bigwedge_{0<i<n} g(t_i) \bowtie 0$, there exists $i \in [1, n]$ such that $t_b \in (t_{i-1}, t_i)$.

Let us consider first the cases in which $\bowtie = \geq$ or $i \in [2, n-1]$ or $i = 1$ and $g(t) \bowtie 0$ or $i = n$ and $g(t') \bowtie 0$. Since g is differentiable, for the mean value theorem, there exists a point $t'_b \in (t_{i-1}, t_b)$ such that $\dot{g}(t'_b) = \frac{g(t_b) - g(t_{i-1})}{(t_b - t_{i-1})}$ and therefore $\dot{g}(t'_b) < 0$. Similarly, there exists a point $t''_b \in (t_b, t_i)$ such that $\dot{g}(t''_b) = \frac{g(t_i) - g(t_b)}{(t_i - t_b)}$ and therefore $\dot{g}(t''_b) > 0$. Thus, $\dot{g}$ is not constant over (t_{i-1}, t_i) contradicting the hypothesis.

Let us now consider the case in which $\bowtie = >$, $i = 1$ and $g(t) = 0$ (the case $i = n$ and $g(t') = 0$ is similar). By hypothesis, $\dot{g}(t) > 0$. Thus, there exists

$t_0 \in (t_{i-1}, t_b)$ such that $g(t_0) > 0$. As before there exists a point $t'_b \in (t_0, t_b)$ such that $\dot{g}(t'_b) = \frac{g(t_b)-g(t_0)}{(t_b-t_0)}$ and therefore $\dot{g}(t'_b) < 0$. Similarly, there exists a point $t''_b \in (t_b, t_i)$ such that $\dot{g}(t''_b) = \frac{g(t_i)-g(t_b)}{(t_i-t_b)}$ and therefore $\dot{g}(t''_b) > 0$. Therefore $\dot{g}$ is not constant over (t_{i-1}, t_i) contradicting the hypothesis.

We conclude that $\forall \epsilon \in (t, t'), g(\epsilon) \bowtie 0$. $\diamond$

Hereafter, unless otherwise specified, we assume that every universal quantifier occurs positively in $Trans_S$ and that it is in the form $\forall \epsilon \in [t, t'], g(\epsilon) \bowtie 0$, with $\bowtie \in \{<, \leq, >, \geq, =\}$. As shown by the previous theorem, the assumption does not limit the approach, but it simplifies the presentation of the results of this section.

The definition of *Constant*() contains quantified sub-formulas in the form $\forall \epsilon \in [t, t'], \dot{g} \bowtie 0$. Therefore, the reduction can be iterated trying to remove the quantifiers.

Theorem 3 can be used to simplify the encoding of the invariant of an HS. Let the invariant be in the form $g(X) \bowtie 0$ ($\bowtie \in \{\geq, \leq, >, <, =\}$) and $f : \mathbb{R} \to \mathbb{R}^{|X|}$ be the solution of the flow condition. If f and g are differentiable functions and $\frac{d}{dt}(g \circ f) \bowtie 0$ has finite variability, then $\forall \epsilon \in [t, t'], g(f(\epsilon)) \bowtie 0$ iff there exists a finite sequence of real numbers $t_0 = t < \ldots < t_n = t'$ such that $\bigwedge_{0 \leq i \leq n} g(f(t_i)) \bowtie 0 \wedge \bigwedge_{0 < i \leq n} Constant(\frac{d}{dt}(g \circ f) \geq 0, t_{i-1}, t_i)$.

The geometrical interpretation of $\frac{d}{dt}(g \circ f)$ is the scalar product of the gradient of the curve g and the derivative vector $\dot{f}$: in fact, $\frac{d}{dt} g(f(t)) = \bigtriangledown g \cdot \dot{f}$ where $\bigtriangledown g = \langle \frac{\partial g}{\partial x_1}, \ldots, \frac{\partial g}{\partial x_n} \rangle$. Therefore, in the theorem, the condition of $\dot{g} \geq 0$ of being constant in the interval means that the function f is uniformly getting closer to (or farther from) the curve g in that interval.

As a side note, in the case of ODEs $\dot{X} = F(X)$, the new quantified formula $\forall \epsilon \in [t, t'], \frac{d}{dt}(g \circ f) \geq 0$ is equivalent to the invariant $\bigtriangledown g \cdot F \geq 0$. Thus, the reduction can be also applied without need of the primitive solutions.

In the case that the invariants are polynomial and the continuous variables are polynomial functions of time, the derivative will eventually reduce to zero.

3.2.4 Applications

Application to polynomial hybrid automata

We consider the class of HS where the invariants and the primitive solution of the ODEs are polynomial functions of time (see also [Frä01]). The polynomial may contain some discrete variables as coefficients to account for uncertainties in the inputs, model parameters, etc. Note that several classes of HS with linear ODE can be expressed as a polynomial hybrid automaton, since the solution to the linear system of ODEs can be expressed as a quantifier free formula in the theory of reals [LPY01]. We describe a quantifier-free encoding for some classes of linear hybrid systems in Section 3.4.

Theorem 5 *The invariant of a polynomial hybrid automaton can be encoded with a quantifier-free formula.*

Proof. In the case of polynomial hybrid automata, the invariant $g \bowtie 0$ is encoded into a formula in the form $\forall \epsilon \in [t, t'], g(f(\epsilon)) \bowtie 0$. If g and f are polynomials, $g \circ f$ is also a polynomial. The derivative of a polynomial has a lower degree than the polynomial itself. Thus, at every application of Theorem 3, the degree of the polynomial inside the quantifier strictly decreases. Thus, after a finite number of applications of the theorem, we obtain a quantifier-free formula. $\diamond$

Example 4 *Let us consider the classical example of the bouncing ball. Suppose the ball moves in two dimensions x and y, where x is the horizontal coordinate, with $\dot{x} = v_0$, and y is the vertical coordinate, with $\dot{y} = w$ and $\dot{w} = -g$. Thus, the primitive solution is $x(t) = v_0 t + x_0$,*

$y(t) = -\frac{g}{2}t^2 + w_0 t + y_0$, *and* $w(t) = -gt + w_0$. *Suppose the ball is bouncing on a parabolic hill, a curved surface with equation* $y + ax^2 + bx + c = 0$. *The invariant of the continuous transition is* $y + ax^2 + bx + c \geq 0$ *and its encoding is* $\forall \epsilon \in [t, t'], y(\epsilon) + ax^2(\epsilon) + bx(\epsilon) + c \geq 0$, *which is quadratic in* ϵ. *After applying the Theorem 3 twice, we obtain the following quantifier-free formula:* $y(t) + ax^2(t) + bx(t) + c \geq 0 \wedge$
$y(t_1) + ax^2(t_1) + bx(t_1) + c \geq 0 \wedge$
$y(t') + ax^2(t') + bx(t') + c \geq 0 \wedge$
$((w(t) + 2av_0x(t) + bv_0 \geq 0 \wedge w(t_1) + 2av_0x(t_1) + bv_0 \geq 0) \vee$
$(w(t) + 2av_0x(t) + bv_0 \leq 0 \wedge w(t_1) + 2av_0x(t_1) + bv_0 \leq 0)) \wedge$
$((w(t_1) + 2av_0x(t_1) + bv_0 \geq 0 \wedge w(t') + 2av_0x(t') + bv_0 \geq 0) \vee$
$(w(t_1) + 2av_0x(t_1) + bv_0 \leq 0 \wedge w(t') + 2av_0x(t') + bv_0 \leq 0))$

Application to non-linear hybrid automata

In the general case of non-linear hybrid automata (here meant as hybrid systems with non-polynomial functions), the reduction of Theorem 3 may result in more complex quantified formulas. Even if we restrict to polynomial invariants, their composition with transcendental primitive solutions may yield complex derivatives. However, in many cases, we can convert the derived quantified formula into a polynomial which is simpler than the original[3].

Example 5 *Let us consider a temperature controller. The system is parameterized by the lower and upper temperature limits* m *and* M*, the outside temperature* u*, the rate* b *of temperature exchanged with the outside, and the rate* c *of temperature increase due to the heater. The constraints on the parameters are* $u < m < M \wedge c > 0 \wedge b > 0$. *The hybrid automaton is defined as follows:*

[3]This conversion is not currently automated.

- $V = \{h\}$ *where* h *is a variable representing the heater.*
- $X = \{x\}$ *where* x *represents the temperature.*
- $Init := m \leq x \leq M$.
- $Invar := (h = 0 \rightarrow x \geq m) \wedge (h = c \rightarrow x \leq M)$.
- $Trans := (h = 0 \rightarrow (x = m \wedge h' = c)) \wedge (h = c \rightarrow (x = M \wedge h' = 0)) \wedge x' = x$.
- $Flow := \dot{x} = b(u - x) + h$.

The primitive solution of the ODE when the location is $h = c$ *is* $x(t) := u + \frac{(x(0)-u)}{b}e^{(-b*t)} + \frac{c}{b}$. *Its derivative is* $x(t) := -(x(0) - u)e^{(-b*t)}$, *which never changes sign. Therefore, applying Theorem 3,* $\forall \epsilon \in [t, t'], x \geq m$ *is translated into the formula* $x(t) \geq m \wedge x(t') \geq m$ *and similarly for* $\forall \epsilon \in [t, t'], x \leq M$.

Example 6 *Consider the roundabout collision avoidance system example (cfr. e.g. [PC09]). The continuous dynamics of a safe circular maneuver is described by the following equations* $\dot{x_1} = d_1, \dot{x_2} = d_2, \dot{d_1} = -\omega d_2, \dot{d_2} = \omega d_1, \dot{y_1} = e_1, \dot{y_2} = e_2, \dot{e_1} = -\rho e_2, \dot{e_2} = \rho e_1, (x_1 - y_1)^2 + (x_2 - y_2)^2 \geq p^2$.

The primitive solution of the differential equations is:

$$\begin{aligned} x_1 = \frac{1}{\omega}sin(\theta),\ x_2 = -\frac{1}{\omega}cos(\theta), \\ d_1 = cos(\theta),\ d_2 = sin(\theta),\ \theta = \omega t + t_0, \\ y_1 = \frac{1}{\rho}sin(\xi),\ y_2 = -\frac{1}{\rho}cos(\xi), \\ e_1 = cos(\xi),\ e_2 = sin(\xi),\ \xi = \rho t + t_0 \end{aligned}$$

Substituting the primitive solution into the invariant $(x_1 - y_1)^2 + (x_2 - y_2)^2 \geq p^2$ *we obtain the formula:*

$$\frac{1}{\omega^2} + \frac{1}{\rho^2} - \frac{2}{\omega\rho}sin(\theta)sin(\xi) - \frac{2}{\omega\rho}cos(\theta)cos(\xi) \geq p^2.$$

which can be rewritten into: $\phi := \frac{1}{\omega^2} + \frac{1}{\rho^2} - \frac{2}{\omega\rho} cos(\theta - \xi) \geq p^2$.

The standard quantified encoding is $\forall t \in [0, \delta], \phi(t)$. *Applying Theorem 3, we obtain the formula:*

$$\begin{aligned} \phi(0) \wedge \phi(\delta) \wedge \quad & (\forall t(-sin(\theta - \xi)(\omega - \rho) \geq 0) \vee \\ & \forall t(-sin(\theta - \xi)(\omega - \rho) \leq 0)). \end{aligned}$$

The quantified sub-formulas can be rewritten into polynomials over θ *and* ξ. *For example,* $\forall t(-sin(\theta - \xi)(\omega - \rho) \geq 0)$ *can be rewritten into* $\forall t(\omega - \xi \geq 0 \wedge (\pi \leq \theta - \rho \leq 2\pi) \vee \omega - \xi \leq 0 \wedge (0 \leq \theta - \rho \leq \pi))$. *Since* θ *and* ρ *are linear, this can be converted into an equivalent quantifier-free one.*

Example 7 *Consider the steering car example of [IUH11]. The flow invariant of the location* correct_left *is* $\dot{p} = -r * sin(\gamma), \dot{\gamma} = \omega, \dot{c} = -2, -1 \leq p \leq 1, c \geq 0$. *Let us make the example more complex (and realistic) considering an (uniformly) accelerated rotation by adding* $\dot{\omega} = \alpha, 0 \leq \alpha \leq 3$, *where* α *is a (real valued) discrete variable.*

The semantics of this flow invariant is that during a timed transition of δ *time units starting from the state* $p(0) = p_0, \gamma(0) = \gamma_0, c(0) = c_0, \omega(0) = \omega_0$, *there exist the continuous differentiable functions* p, γ, c, ω *that satisfy the ODEs* $\dot{p} = -r * sin(\gamma), \dot{\gamma} = \omega, \dot{c} = -2, \dot{\omega} = \alpha$ *and such that* $\forall t, 0 \leq t \leq \delta(-1 \leq p(t) \leq 1 \wedge c(t) \geq 0 \wedge 0 \leq \alpha \leq 3)$.

The removal of quantifiers is very similar to the Example 6.

The quantification can be distributed obtaining:

$$\begin{aligned} I_1 &:= \forall t, 0 \leq t \leq \delta(-1 \leq p(t) \leq 1) \\ I_2 &:= \forall t, 0 \leq t \leq \delta(c(t) \geq 0) \\ I_3 &:= \forall t, 0 \leq t \leq \delta(0 \leq \alpha \leq 3) \end{aligned}$$

I_3 *is equivalent to* $0 \leq \alpha_0 \leq 3$ *since* α *does not change during the timed transition.*

I_2 is equivalent to $c(0) \geq 0$ and $c(\delta) \geq 0$ since c is linear. This can be obtained also from Theorem 3 by replacing $\dot{c}$ with -2 and simplifying.

In order to remove the quantification of I_1, we apply the Theorem 3 by obtaining

$$\begin{aligned} I_1 \;\equiv\; & -1 \leq p(0) \leq 1 \wedge -1 \leq p(\delta) \leq 1 \wedge \\ & (\forall t, 0 \leq t \leq \delta(\dot{p} \geq 0) \vee \forall t, 0 \leq t \leq \delta(\dot{p} \leq 0)) \end{aligned}$$

By replacing $\dot{p}$ with $-r * sin\gamma$ we obtain the invariant condition:

$$I_{1a} \;:=\; \forall t, 0 \leq t \leq \delta(-r * sin(\gamma) \geq 0)$$

This can be solved considering $0 \leq \gamma \leq 2\pi$ by taking $\pi \leq \gamma \leq 2\pi$. This results in the invariant condition:

$$I_{1b} \;:=\; \forall t, 0 \leq t \leq \delta(\pi \leq \gamma(t) \leq 2\pi)$$

Applying again Theorem 3, we obtain an equivalent formula containing $\forall t, 0 \leq t \leq \delta(\dot{\gamma} \geq 0)$. Now, since γ is linear we can remove the quantification in the standard way.

3.3 Encoding of systems with Polynomial Dynamics

In this section, we show how Theorem 3 can be exploited to automatically encode a HS with polynomial dynamics into a transition system with quantifier-free formulas.

Theorem 3 states the existence of the points $t_1, \ldots, t_n$ where the derivative changes sign. However, such points are unknown. The encoding of a HS into a transition system must thus implicitly represent when the derivative of the invariant changes sign. This is achieved by simply forcing that the sign of the derivative is constant throughout the timed transition. The encoding implicitly concatenates timed transitions one after the other, delegating to the search the task of finding the sequence of time points that split the interval, so that the sign of the derivative is uniformly constant in the resulting trace.

Given a formula T including the invariant condition $\forall \epsilon \in [t, t'], g(\epsilon) \bowtie 0$, the condition can be locally replaced with $g(t) \bowtie 0 \wedge g(t') \bowtie 0 \wedge \mathit{Constant}(\dot{g}, t, t')$ obtaining a new formula $\tau(T)$.

τ performs a recursive substitution of the quantified expressions. The recursion terminates when the quantified formula is a linear polynomial, thus allowing us to trivially remove the quantifiers. τ is defined recursively as follows:

$$\begin{aligned} \tau(\psi_1 \wedge \psi_2) &:= \tau(\psi_1) \wedge \tau(\psi_2) \\ \tau(\psi_1 \vee \psi_2) &:= \tau(\psi_1) \vee \tau(\psi_2) \\ \tau(\neg\psi) &:= \neg\psi, \ (\psi \text{ is a predicate}) \\ \tau(\forall \epsilon \in [t, t'], g(\epsilon) \bowtie 0) &:= \begin{cases} g(t) \bowtie 0 \wedge g(t') \bowtie 0 & \text{if } g \text{ linear} \\ g(t) \bowtie 0 \wedge g(t') \bowtie 0 \wedge & \\ \tau(\mathit{Constant}(\dot{g}, t, t')) & \text{otherwise} \end{cases} \end{aligned} \tag{3.1}$$

The correctness of the transformation is given by the following theorem.

Theorem 6 *If S_H is the encoding of the HS H and $\tau(S_H)$ is the transition system obtained by replacing $Trans$ with $\tau(Trans)$, then $\tau(S_H)$ is the encoding of a sampling refinement of H.*

Proof. ($\Leftarrow$) If a sequence of states satisfies $\tau(S_H)$, then by Theorem 3, the sequence satisfies also S_H, and by Theorem 1, it represents a path of H.

($\Rightarrow$) Consider a hybrid trace $\langle f_0, I_0\rangle, \langle f_1, I_1\rangle, \ldots, \langle f_k, I_k\rangle$ which is a path of H. Assuming that $\dot{g}$ has finite variability, we can refine the hybrid trace into a new hybrid trace in which $\dot{g}$ is constant in every interval. The new hybrid trace also satisfies H by Theorem 2 and thus the corresponding discrete path $s_0; \ldots ; s_k$ satisfies its encoding S_H. At every i, if s_i satisfies $\forall \epsilon \in [t, t'], g(\epsilon) \bowtie 0$, then both $f(s_i, t)$ and $f(s_i, t')$ satisfy $g \bowtie 0$. Since $\dot{g}$ has constant sign in I_i, s_i satisfies also $\tau(Trans)$. Therefore the discrete path satisfies also $\tau(S_H)$. $\diamond$

The recursive definition of τ in (3.1) creates a formula whose size is exponential in the degree of the polynomial inside the invariant. We use the following equivalence to keep the size of the encoding *linear* in the degree of the polynomial (here g is not linear):

$$\begin{aligned}
\tau(\mathit{Constant}(g, t, t')) &= (g(t) \geq 0 \wedge g(t') \geq 0 \wedge \tau(\mathit{Constant}(\dot{g}, t, t'))) \vee \\
&\quad\; (g(t) \leq 0 \wedge g(t') \leq 0 \wedge \tau(\mathit{Constant}(\dot{g}, t, t'))) \\
&= ((g(t) \geq 0 \wedge g(t') \geq 0) \vee (g(t) \leq 0 \wedge g(t') \leq 0)) \wedge \\
&\quad\; \tau(\mathit{Constant}(\dot{g}, t, t'))
\end{aligned}$$

The sequential encoding may force the split of a continuous transition in several transitions, since the predicates introduced to remove the quantifiers force the derivatives of the invariant conditions to be constant. While the encoding enables to remove the quantifier, the depth of the bounded model checking formula may increase due to the splitting. In incremental bounded model checking, the burden of finding how many splits are necessary is delegated to the search.

In the case of polynomial hybrid automata we can compute an upper bound on the number of consecutive continuous transitions (continuous transitions not separated by a discrete transition) needed to simulate the longest quantified continuous transition (the continuous transition with the maximum time elapse).

We can compute the upper bound on the number of intervals needed to "cover" the quantified continuous transition for the invariant predicate $\forall \epsilon \in [t, t'], g(\epsilon) \bowtie 0$. If $\Omega(g)$ is the degree of the polynomial, then the maximum number of intervals that have to be considered is $ub(g) = \frac{\Omega(g) * (\Omega(g) - 1)}{2}$. In fact, the i-th derivative of g has degree $\Omega(g) - i$ and thus changes sign $\Omega(g) - i$ times.

3.4 Encoding of systems with Linear Dynamics

In this section we describe how we can encode sub-classes of hybrid systems that have a *linear dynamic* (i.e. the flow condition in each location is defined by a system of linear ODEs) using quantifier free formulas. Moreover, we consider systems where the initial condition, the invariant condition and the transition relation are expressed in $\mathcal{T}(\mathbb{R})$. The resulting encoding will be in the theory of reals ($\mathcal{T}(\mathbb{R})$). We rely on the results presented by [LPY01], which show how the primitive solution of several classes of systems with linear dynamics can be encoded in $\mathcal{T}(\mathbb{R})$. While their approach handles invariants, it still relies on universal quantification. We show that in two cases the universal quantifier can be removed by applying Theorem 5. The process is straightforward in one case, while it requires several steps in the other.

In the following, we show the quantifier free encoding considering a single location $q \in Q$ to the automaton. The flow condition in q is of the form $\dot{\vec{x}} = A\vec{x} + b$, where $A \in \mathbb{R}^{n \times n}, b : \mathbb{R} \to \mathbb{R}^n$. Given a matrix $M \in \mathbb{R}^q \times \mathbb{R}^s$, with $q, s \in \mathbb{N}$, we will write M_{ij} to refer to the element at the i-th row and at the j-th column of M. We will avoid one index in the case of vectors, where $s = 1$. Also, we denote with Λ the set of eigenvalues of A. To ease the presentation, we will use the symbol δ to represent the amount of time $t' - t$ elapsed in a continuous transition.

Let L be a set of symbolic parameters that can be used in the inputs b. The value of each $l \in L$ is unknown, but it does not change during the execution of the system. Moreover, the set L is disjoint from the set of the variables of the hybrid system ($X \cup V$). Let P be a set of functions over

δ (e.g. $P = \{p(\delta) = \delta^n, n \in \mathbb{N}\}$ is the set of all the powers of δ with a natural exponent). Let M_P be a set of inputs parameterized by P:

$$M_P := \Big\{ b \in [b_1, \ldots, b_n]^T \mid \text{for } i \in [1,n], b_i(\delta) = \sum_{l=1}^{r} u_{i,l} p_l(\delta), \\ p_l(\delta) \in P, u_{i,l} \text{ is a } \Sigma_{\mathbb{R}}\text{-formula over } L \Big\}$$

P determines the function of the terms $p_l(\delta)$. We will consider different families of inputs, parameterized by different families of functions P.

Given a linear system $\dot{\vec{x}} = A\vec{x} + b$ the reachability problem can be expressed in the theory of reals if the matrix A and the inputs b have a particular structure[LPY01]:

- A is nilpotent and $P = \{\delta^n, n \in \mathbb{Z}\}$,
- A is diagonalizable, all its eigenvalues are real and $P = \{e^{u_l \delta}, u_l \notin \Lambda, u_l \in \mathbb{Q}\}$,
- A is diagonalizable, all its eigenvalues are imaginary and $P = \{\sin(u_l\delta), u_l \notin \Lambda, u_l \in \mathbb{Q}\} \cup \{\cos(u_l\delta), u_l \notin \Lambda, u_l \in \mathbb{Q}\}$.

We will provide a quantifier-free encoding for the first two cases.

Note that we do not consider symbolic coefficients in the matrix A. While in the first case obtaining an exact solution in the theory of reals is straightforward, also in the presence of symbolic coefficients of the matrix, the second case is more involved and, since its applicability depends on the coefficient of A (i.e. eigenvalues and diagonalizability), it requires imposing several constraints on the parameters.

The general solution of $\dot{\vec{x}} = A\vec{x} + b$ is:

$$\vec{x}(\delta) = f(\vec{x}, \delta, b) = e^{A\delta} X + \Psi(\delta, b)$$

where:

$$\Psi(\delta, b) = \int_{s=0}^{\delta} e^{A(\delta - s)} b(s) \, \mathrm{d}s \qquad e^{A\delta} = \sum_{k=0}^{\infty} \frac{\delta^k}{k!} A^k$$

3.4.1 Reduction in the nilpotent case

Suppose that the flow condition in the location of a hybrid automaton is of the form $\dot{\vec{x}} = A\vec{x} + b$, where A is nilpotent and the inputs are of the form $P = \{\delta^i, i \in \mathbb{N}\}$. Also, suppose that $g(X, \delta) \bowtie 0$ be the invariant in the location, and $g(X, \delta)$ be a multivariate polynomial with variables in X and $P = \{\delta^i, i \in \mathbb{N}\}$.

In this case, for each $1 \leq i \leq n$ the solution is:

$$f(\vec{x}, \delta, b)_i := \sum_{k=1}^{n} \gamma_{ij}(X) + \delta^{k-1} + \sum_{k=0}^{v} \rho_{i,k}(b)\delta^{k-1}$$

for some $v \in \mathbb{N}$, some polynomials $\rho_{i,k}(b)$ and $\gamma_{ij}(X) = \sum_{j=1}^{n} (A_k)_{ij} x_j \frac{1}{k!}$. Note that we do not know a priori $\sum_{k=0}^{v} \rho_{i,k}(b)\delta^k$, since it depends on the inputs b. However, an expression of this form can always be obtained as a solution of the integral in $\Psi(\delta, b)$ (i.e. in this case we are just integrating a polynomial over t). Thus, the primitive solution $f(\vec{x}, \delta, b)$ is a $\Sigma_{\mathbb{R}}$-formula without transcendental functions (i.e. a polynomial).

Also the invariant $g(X, \delta)$ is a multivariate polynomial over X and $P = \{\delta^i, i \in \mathbb{N}\}$. The encoding of the continuous transition in the location is:

$$\delta > 0 \wedge \bigwedge_{i}^{n} x_i' = f(\vec{x}, \delta, b)_i \wedge \forall \epsilon \in [0, \delta], g(X, \epsilon) \bowtie 0$$

We are in the case of polynomial hybrid automata, so we apply the Theorem 5 to obtain a quantifier-free encoding.

Note that the bouncing ball example (see Example 4) with a set of instantiated parameters, falls in this class of systems.

Example 8 *The movement of two vehicles which follows a uniformly accelerated motion can be modeled with the linear dynamic* $\dot{\vec{x}} = A\vec{x} + b$

where: $X^T := \begin{bmatrix} x_1 & v_1 & a_1 & x_2 & v_2 & a_2 \end{bmatrix}$. $A := \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ *and* $b^T := \begin{bmatrix} 0 & 0 & 10 & 0 & 0 & 6 \end{bmatrix}$. *The invariant of the system* $x_1 \leq x_2 + s_d$ *guarantees that the first vehicle follows the second vehicle respecting a safety distance* s_d. *The primitive solutions are:* $x_1(\delta) = 5\delta^2 + v_1(0)t + x_1(0)$, $v_1(\delta) = 10\delta + v_1(0)$, $a_1(\delta) = 10$, $x_2(\delta) = 3\delta^2 + v_2(0) + x_2(0)$, $v_2(\delta) = 6\delta + v_2(0)$, $a_2(\delta) = 6$. *The quantified invariant may be rewritten as:*

$$\forall \epsilon \in [0, \delta], 2\epsilon^2 + (v_1(0) - v_2(0))\epsilon + x_1(0) - x_2(0) - s_d \leq 0$$

Thus, applying Theorem 5 we remove the quantifiers:

$$\begin{aligned} x_1(0) - x_2(0) - s_d \leq 0 \wedge \\ 2\delta^2 + (v_1(0) - v_2(0))\delta + x_1(0) - x_2(0) - s_d \leq 0 \wedge \\ (v_1(0) + v_1(0) \geq 0 \wedge 4 * \delta + v_1(0) + v_1(0) \geq 0) \wedge \\ (v_1(0) + v_1(0) < 0 \wedge 4 * \delta + v_1(0) + v_1(0) < 0) \end{aligned}$$

3.4.2 Reduction in the case A is diagonalizable with real eigenvalues

Suppose that the flow condition in the location of a hybrid automaton is of the form $\vec{\dot{x}} = A\vec{x} + b$, where A is diagonalizable, all its eigenvalues are real (i.e. $\Lambda \subset \mathbb{R}$) and the inputs are of the form $P = \{e^{u_l \delta}, u_l \notin \Lambda, u_l \in \mathbb{Q}\}$. Also, suppose that $g(X, \delta) \bowtie 0$ is the invariant in the location, and $g(X, \delta)$ is a multivariate polynomial with variables in X and $P = \{e^{u_l \delta}, u_l \notin \Lambda, u_l \in$

$\mathbb{Q}\}$ [4]. Since A is diagonalizable, there exists an invertible matrix $T \in \mathbb{R}^{n\times n}$ such that $A = TDT^{-1}$, where $D = \begin{bmatrix} \lambda_1 & & \\ & \dots & \\ & & \lambda_n \end{bmatrix}$ is the diagonal matrix of A. Hence, we can compute $e^{A\delta} = e^{TDT^{-1}\delta} = T \begin{bmatrix} e^{\lambda_1\delta} & & \\ & \dots & \\ & & e^{\lambda_n\delta} \end{bmatrix} T^{-1}$.

In this case, for each $1 \leq i \leq n$ we have:

$$f(\vec{x}, \delta, b)_i := \sum_{k=1}^{n} \gamma_{i,k}(x)e^{\lambda_k\delta} + \sum_{k=1}^{s} \psi_{ik}(b)e^{\nu_k\delta}$$

where $s \in \mathbb{N}$, $\gamma_{i,k}(x)$ is a polynomial, and for all $1 \leq k \leq s$, $\nu_k \in \mathbb{Q}$ and $\psi_{ik}(b)$ is a polynomial. Let us write $f(\vec{x}, \delta, b)_i$ as follows (for some natural $q > 0$):

$$f(\vec{x}, \delta, b)_i := \sum_{k=1}^{q} \phi_{ik}(X, b)e^{\eta_k\delta}$$

The formulas $f(\vec{x}, \delta, b)_i$ are such that δ occurs only in the exponent of e, and hence we have terms like $e^{\eta\delta}$, where $\eta \in \mathbb{Q}$. We also requires the same property in the invariant g. Thus, we substitute the exponential with a new variable z both in the solution and in the invariant as follows:

- We compute a common denominator d among all the rational coefficients η (i.e. $d = \prod_{\textit{for all the coefficients } \eta} den(\eta)$), where $den(\eta)$ is the denominator of η.
- We define the variable $z = e^{\frac{\delta}{d}}$.
- We substitute $e^{\frac{\delta}{d}}$ with z in the solution and in the invariant:
  - for all $1 \leq i \leq n$, $\hat{f}(\vec{x}, z, b)_i := f(\vec{x}, \delta, b)[z/e^{\frac{\delta}{d}}]_i)$.

[4] Note that u_l cannot be an eigenvalue of the system. This condition is necessary to get a solution where δ appears only as exponent of e, thus enabling the removal of the exponential function via substitution.

  – $\hat{g}(X, z) := g(X, \delta)[z/e^{\frac{\delta}{d}}]$.

- If $\hat{f}$ contains a negative power of z, i.e., some z^{-l} with $l > 0$, we substitute it with w^l, where $zw = 1$:
  – for all $1 \leq i \leq n$, $\hat{f}(\vec{x}, z, w, b) = \hat{f}(\vec{x}, z, b)[w^l/z^{-l}]_i \wedge wz = 1$.

Remark 3 *If we want to obtain a polynomial, the substitution of $e^{\frac{\delta}{d}}$ with z forbids the use of the variable t, which tracks the total amount of elapsed time, in the transition system. In fact, t is updated in the continuous transition using a logarithm ($t' = \ln(z) - t$). This shows that in the system we cannot have continuous variables that evolve as clocks (i.e. a variable x such that $\dot{x} = 1$).*

The encoding of the continuous transition in the location is:

$$\delta > 0 \wedge \bigwedge_i^n x'_i = f(\vec{x}, \delta, b)_i \wedge \forall \epsilon \in [0, \delta], g(\epsilon) \bowtie 0$$

Performing the substitution we obtain the following formula:

$$z > 1 \wedge zw = 1 \wedge \bigwedge_i^n x'_i = \hat{f}(\vec{x}, z, w, b)_i \wedge \forall z_\epsilon \in [1, z], \hat{g}(X, z_\epsilon) \bowtie 0$$

Note that $\hat{g}(X, z_\epsilon)$ may contain negative powers of z. However, $z > 0$ and we can multiply both sides of $\hat{g}(X, z_\epsilon)$ by z^l where l is the greatest negative order with which occurs in $\hat{g}$. Now we can recursively apply Theorem 3 to obtain a quantifier free formula.

Example 9 *Consider a system with $X^T = [x, y]$, $A = \begin{bmatrix} 1 & \frac{-12}{5} \\ \frac{-12}{5} & 1 \end{bmatrix}$, $b^T = [1, 1]$ and invariant $x^2 \geq 0$. The eigenvalues of A are $\frac{-7}{5}$ and $\frac{17}{5}$ and A is diagonalizable. The solution $X(\delta)$ is:*

$$x(\delta) := \frac{x - y}{2} e^{\frac{17}{5}\delta} + \frac{x + y}{2} e^{-\frac{7}{5}\delta} - \frac{5}{7} e^{-\frac{7}{5}\delta} + \frac{5}{7}$$
$$y(\delta) := \frac{y - x}{2} e^{\frac{17}{5}\delta} + \frac{x + y}{2} e^{-\frac{7}{5}\delta} - \frac{5}{7} e^{-\frac{7}{5}\delta} + \frac{5}{7}$$

In this case we use the following variables for the substitution (note that the $d = 5$): $z = e^{\frac{1}{5}\delta}$:

$$\hat{x}(z) := \frac{x-y}{2}z^{17} + \frac{x+y}{2}z^{-7} - \frac{5}{7}z^{-7} + \frac{5}{7}$$
$$\hat{y}(z) := \frac{y-x}{2}z^{17} + \frac{x+y}{2}z^{-7} - \frac{5}{7}z^{-7} + \frac{5}{7}$$

$\hat{g}(z)$ is:

$$\begin{aligned}\hat{g}(z) :=&(\frac{5}{7}x + \frac{5}{7}y + \frac{x+y}{2} + \frac{25}{49})z^{-14} + (\frac{5}{7}x + \frac{5}{7}y + \frac{50}{49})z^{-7} + \\ &(\frac{x^2}{2} + \frac{5}{7}x - \frac{y^2}{2} - \frac{5}{7}y)z^{-7}z^{17} + \frac{x-y^2}{2}z^{34}(\frac{5}{7}x - \frac{5}{7}y)z^{17} + \frac{25}{49}\end{aligned}$$

Then, we multiply both sides of $\hat{g}(z) \bowtie 0$ by z^{14}:

$$\begin{aligned}&\frac{x-y^2}{2}z^{48} + (\frac{5}{7}x - \frac{5}{7}y)z^{31} + (\frac{x^2}{2} + \frac{5}{7}x - \frac{y^2}{2} - \frac{5}{7}y)z^{24} + \frac{25}{49}z^{14} \\ &\qquad (\frac{5}{7}x + \frac{5}{7}y + \frac{50}{49})z^{7} + (\frac{5}{7}x + \frac{5}{7}y + \frac{x+y}{2} + \frac{25}{49}) \bowtie 0\end{aligned}$$

3.5 Related work

Several works focus on the problem of encoding hybrid systems into transition systems, but they use less expressive invariants or they restrict the class of the analyzed hybrid automata. There are examples of encodings into timed automata [NMA$^+$02, ACKS02, Sor02, dMRS02] and linear hybrid automata [dMRS03, ABCS05, ÁBKS05]. All these works do not consider the problem of quantified invariants since for linear hybrid automata (and thus for timed automata) if the invariant holds in the first and in the last instant of a timed transition, then the invariant holds also in all the intermediate time points, thus resulting in a quantifier-free encoding.

Other approaches [BZL10, ERNF11, IUH11, GKC13] focus on non-linear hybrid automata.

In [BZL10], the authors solve the reachability problem for non-linear convex hybrid automata. The restriction to convex invariant and linear flow conditions, or to monotonic invariant and convex flow, allows an easy encoding of invariants without quantifiers. Many examples, including those mentioned in this chapter, do not fall in this class of automata.

All the approaches [ERNF11, IUH11, GKC13, PKV13] defines and use decision procedures that can directly handle ODEs. In [ERNF11] the authors propose an SMT solver modulo ODEs, that can be used to perform bounded model checking on hybrid automata where the flow conditions are ODEs. The only allowed invariants are of the form $x \in [l, u]$, where x is a continuous variable and $l, u \in \mathbb{R}$. Their main focus is on the integration of numerical methods to compute the initial value problem for ODEs, while they cannot manage more complex invariants (e.g. linear functions). The authors of [IUH11] compute the precise intersection of the continuous flow with the guards of the hybrid automaton. The solver can in principle handle invariants, but the authors state that the implementation is not mature enough to evaluate the approach. Recently, the authors of [GKC13] propose a general SMT-theory that can directly handle Ordinary Differential Equations (ODE). Since the satisfiability problem for theories that admit transcendental functions and ODEs is not decidable, the proposed decision procedure considers the δ-satisfiability problem, where a formula is δ-satisfiable if under some δ-perturbations a syntactic variant of the formula is satisfiable. The proposed framework allow to handle the universal quantifiers of the invariants natively. Our approach handles hybrid systems with simpler dynamics but precisely encodes the behavior of the original system, without considering arbitrary perturbations. However, in real world application it is an advantage to verify a systems considering also possible small perturbation of its behavior, since this would also certify its robustness. Other approaches based on motion planning [PKV13] do not encode symbolically the invariants, since they simulate the ODEs using numerical methods (without considering numerical errors). In contrast, we encode a set of continuous transitions.

The quantifier-free encoding that we propose is related to quantifier elimination procedures (see, e.g., [Col75, DSW98]). It is not a quantifier elimination procedure in that it contains new variables that are implicitly existentially quantified. In fact, we apply the reduction even in some cases of transcendental functions. The burden to remove the quantifiers is delegated to the verification techniques if necessary. We claim that quantifier elimination is somehow an overkill: the verification techniques does not often need the precise region of points where the invariant holds; it is usually sufficient either to pick some "good" values (in case of reachability) or to find "good" invariants (in case of safety verification).

The prominent approaches to the verification of hybrid systems are based either on the exploration of the reachable states or on deductive systems. We refer the readers to [Alu11] for a recent survey. The focus of our work is on the SMT-based paradigm, which, although less mature, seems promising.

Our settings also differs from the works that build abstractions for HSs. The approaches described in [Tiw08, ST11a] use techniques based on the sign of derivatives such as ours. However, the purpose is different in that they generate over-approximations of the HS.

Finally, we mention the "clock translation" described in [HHWT98], where invariants are translated into constraints on time. However, the translation is restricted to monotonic flows (plus other restrictions on the independence of variables).

Chapter 4

Encoding of Hybrid Automata Network

In this chapter we show the transition system encoding of a *Hybrid Automata Network*. We show the encoding of two different semantics, the *Global-time* [Hen96] and *Local-time* semantics [BJLY98].

The global-time semantic encoding captures the standard semantic of a network of hybrid automata, since it forces the synchronization of the time-elapse steps of all the automata in the network. Thus, the time elapses in the same way in all the automata of the network.

Instead, in the local-time semantic encoding each automaton keeps the amount of time elapsed in a local clock variable that is incremented independently by each automaton: this way, time evolves differently in each component. Then, the encoding forces that all the automata that synchronize must agree on the value of their local clocks, enforcing that the synchronization happened at the same time. Moreover, the same condition on clocks is required at the end of a run. The local-time semantic allows us to investigate alternative, and more efficient, encodings for the reachability problem (Chapter 6) and the verification of scenario specifications (Chapter 7).

In the rest of the chapter, we consider a network of hybrid automata

$\mathcal{N} = H_1||\ldots||H_n$, where for all $1 \leq i \leq n$ each hybrid automaton H_i is defined as $\langle V_i, X_i, \varepsilon_i, Init_i, Invar_i, Trans_i, Flow_i\rangle$. We also assume that all the variables of the automata in the network are disjoint (i.e. for all $1 \leq i < j, V_i \cap V_j = X_i \cap X_j = \emptyset$). While other formalism are available (See e.g.[LSV03]), the local-time semantic encoding strongly rely on this assumption.

4.1 Global-time semantic

We associate to a network $\mathcal{N} = H_1||\ldots||H_n$ a set of first-order transition systems $S_1, \ldots, S_n$, with $S_i = \langle V_{s_i}, \mathcal{W}_{S_i}, Init_{S_i}, Inv_{S_i}, Trans_{S_i}\rangle$, and a synchronization constraint $\textsc{Sync}_{\textsc{GlTime}}$ over the union of the V_{S_i} and $\mathcal{W}_{S_i}$. We define the *global-time semantics* [Hen96] of a network of hybrid automata $\mathcal{N}$ as the symbolic transition system $S_{\textsc{GlTime}}(\mathcal{N}) = \langle V, \mathcal{W}, Init, Inv, Trans\rangle$, where $V := \bigcup_{1\leq i\leq n} V_i$, $\mathcal{W} := \bigcup_{1\leq i\leq n} \mathcal{W}_i$, $Init := \bigwedge_{1\leq i\leq n} Init_{S_i}$, $Inv := \bigwedge_{1\leq i\leq n} Inv_{S_i}$, $Trans := \bigwedge_{1\leq i\leq n} Trans_{S_i} \wedge \textsc{Sync}_{\textsc{GlTime}}$ and for all $1 \leq i \leq n$, S_i is the transition system associated to H_i. S_i is defined as follows:

- $V_{S_i} := V_i \cup X_i$,
- $\mathcal{W}_{S_i} := \{\varepsilon_i, \delta_i\}$, where the domain of ε_i is $A_i \cup \{\textsc{t}, \textsc{s}\}$ and $\delta_i \in \mathbb{R}$,
- $Init_{S_i} := Init_i$,
- $Inv_{S_i} := Invar_i$,
- $Trans_{S_i} := \textsc{Untimed}_i \vee \textsc{Timed}_i \vee \textsc{Stutter}_i$ where
  - $\textsc{Untimed}_i := \bigvee_{a\in A}(\varepsilon_i = a) \wedge Trans_i(V_i, X_i, V_i', X_i')$.
  - $\textsc{Timed}_i := \varepsilon_i = \textsc{t} \wedge \delta_i > 0 \wedge V_i' = V_i \wedge X_i' = f(V_i, \delta_i) \wedge$
    $\forall \epsilon \in [0, \delta_i], Invar_i(V_i, f(V_i, \epsilon))$
  - $\textsc{Stutter}_i := \varepsilon_i = \textsc{s} \wedge V_{S_i}' = V_{S_i}$

The variable δ_i represents the amount of time elapsed during a continuous transition (as noted in the Remark 2). The variable ε selects the events of the automaton. The encoding has two additional event values: $\textsc{t}$ selects the continuous evolution, while $\textsc{s}$ represents the stutter event.

The formula $\textsc{Sync}$ encodes the synchronization of the automata network:

$$
\begin{aligned}
\textsc{Sync}_{\textsc{GlTime}} := \bigwedge_{1\le j<h\le n} \quad & \bigwedge_{a\in A_j\cap A_h} (\varepsilon_j = a \leftrightarrow \varepsilon_h = a) \\
& \wedge(\varepsilon_j = \textsc{t} \leftrightarrow \varepsilon_h = \textsc{t}) \\
& \wedge(\varepsilon_j = \textsc{t} \leftrightarrow \delta_j = \delta_h) \\
& \wedge \bigwedge_{a\in(A_j\setminus A_h)} (\varepsilon_j = a \rightarrow \varepsilon_h = \textsc{s}) \\
& \wedge \bigwedge_{a\in(A_h\setminus A_j)} (\varepsilon_h = a \rightarrow \varepsilon_j = \textsc{s})
\end{aligned}
$$

The condition states, for all the possible couples of automata, that the automata must synchronize when performing transitions that have the same label and on the timed event. Moreover, the constraint forces that the time elapsed on a timed transition must be the same in all the automata (i.e. the time is the same in all the automata in the initial state and it is also the same after the continuous transition, while it cannot change during a discrete transition or during stuttering). Finally $SYNC_{\textsc{GlTime}}$ forces the stuttering of all the automata not involved in the current synchronization.

The stuttering condition can be relaxed in a variant of the $\textsc{Sync}_{\textsc{GlTime}}$ constraint, called *step semantics* [HN03], which allows different automata to execute independent transitions in parallel:

$$\begin{aligned}\textsc{Sync}_{\textsc{GlTime}_{step}} := \bigwedge\nolimits_{1\leq j<h\leq n} & \bigwedge_{a\in A_j\cap A_h} (\varepsilon_j = a \leftrightarrow \varepsilon_h = a)\\ & \wedge(\varepsilon_j = \textsc{t} \leftrightarrow \varepsilon_h = \textsc{t})\\ & \wedge(\varepsilon_j = \textsc{t} \leftrightarrow \delta_j = \delta_h)\end{aligned}$$

4.2 Local-time semantic

We consider the *local-time semantics* of [BJLY98], where time progresses in each component with a local scale by enriching all shared events with time-stamps and synchronizing the components on shared events forcing the time-stamps to be equal. This semantic differs from the standard global-time semantics of [Hen96], where the time event is shared by all components. The two semantics are equivalent from the point of view of reachable states and traces (modulo the partial order over events), but local-time allows a more efficient encoding where the problem is partitioned in sub-problems interfaced by equalities.

We define the semantics of a network of hybrid automata in terms of the symbolic transition system $S_{\textsc{LocTime}}(\mathcal{N})$. As before, we associate to a network $\mathcal{N} = H_1||\ldots||H_n$ a set of first-order transition systems $S_1,\ldots,S_n$, with $S_i = \langle V_{S_i}, \mathcal{W}_{S_i}, Init_{S_i}, Inv_{S_i}, Trans_{S_i}\rangle$, and a synchronization constraint $\textsc{Sync}_{\textsc{LocTime}}$ over the union of the V_{S_i} and $\mathcal{W}_{S_i}$. $S_{\textsc{LocTime}}(\mathcal{N}) = \langle V, \mathcal{W}, Init, Inv, Trans\rangle$ where: $V = \bigcup_i V_{S_i}$; $\mathcal{W} = \bigcup_i \mathcal{W}_{S_i}$; $Init = \bigwedge_i Init_{S_i}$; $Inv = \bigwedge_i Inv_{S_i}$; $Trans = \bigwedge_i Trans_{S_i} \wedge \textsc{Sync}_{\textsc{LocTime}}$.

The elements of S_i are defined as follows:

- $V_{S_i} := V_i \cup X_i \cup \{t_i\}$,
- $\mathcal{W}_{S_i} := \{\varepsilon_i\}$, where the domain of ε_i is $A_i \cup \{\textsc{t}, \textsc{s}\}$,

- $Init_{S_i} := t_i = 0 \wedge Init_i$,
- $Inv_{S_i} := Invar_i$,
- $Trans_{S_i} := \textsc{Untimed}_i \vee \textsc{Timed}_i \vee \textsc{Stutter}_i$ where
  - $\textsc{Untimed}_i := \bigvee_{a \in A}(\varepsilon_i = a) \wedge t'_i = t_i \wedge Trans_i(V_i, X_i, V'_i, X'_i)$.
  - $\textsc{Timed}_i := \varepsilon_i = \textsc{t} \wedge t'_i > t_i \wedge V'_i = V_i \wedge X'_i = f(V_i, t'_i) \wedge$
    $\forall \epsilon \in [t_i, t'_i], Invar_i(V_i, f(V_i, \epsilon))$
  - $\textsc{Stutter}_i := \varepsilon_i = \textsc{s} \wedge V'_{S_i} = V_{S_i} \wedge t'_i = t_i$;

The formula $\textsc{Sync}_{\textsc{LocTime}}$ is defined as follows:

$$\begin{aligned}\textsc{Sync}_{\textsc{LocTime}} := \bigwedge\nolimits_{1 \le j < h \le n} \quad & \bigwedge_{a \in A_j \cap A_h} (\varepsilon_j = a \leftrightarrow \varepsilon_h = a) \wedge (\varepsilon_j = a \rightarrow t_j = t_h) \\ & \wedge \bigwedge_{a \in (A_j \setminus A_h) \cup \{\textsc{t}\}} (\varepsilon_j = a \rightarrow \varepsilon_h = \textsc{s}) \\ & \wedge \bigwedge_{a \in (A_h \setminus A_j) \cup \{\textsc{t}\}} (\varepsilon_h = a \rightarrow \varepsilon_j = \textsc{s})\end{aligned}$$

The local-time semantic encodes the timed transition of each automaton as a local event. This means that in a path of the transition system $S_{\textsc{LocTime}}$ there can exists a state where the local clocks of two automata have a different value (e.g. $t_i \neq t_j$). However, the synchronization constraint force that the time variables of the different automaton must be the same every time the automata synchronize on a shared event.

As in the global-time case, we may have the *step semantics* variant to allow the parallel execution of independent transitions:

$$\textsc{Sync}_{\textsc{LocTime}_{step}} := \bigwedge_{1 \le j < h \le n} \bigwedge_{a \in A_j \cap A_h} (\varepsilon_j = a \leftrightarrow \varepsilon_h = a) \wedge (\varepsilon_j = a \rightarrow t_j = t_h)$$

Note that, since the timed transition is a local action, it can happen in parallel with other discrete transitions.

We say that a state s_k of $S_{\text{LocTime}}(\mathcal{N})$ is *synchronized* iff for $1 \leq i < j \leq n$, $s_k(t_i) = s_k(t_j)$, i.e., the local times are equal. We say that a path $\pi = s_0; a_1; s_1; \ldots; a_k; s_k$ of the FOTS $S_{\text{LocTime}}(\mathcal{N})$ is *synchronized* if s_k is a synchronized state.

The paths of the network $\mathcal{N}$ are given by the synchronized $\mathcal{T}$-paths of $S_{\text{LocTime}}(\mathcal{N})$, where $\mathcal{T}$ is the theory of LRA.

4.3 Local time vs. global time

We denote with $s = \langle s_1, \ldots, s_n \rangle$ a state s of $S_{\text{GlTime}}(\mathcal{N})$ $[S_{\text{LocTime}}(\mathcal{N})]$, where s_i is a state of the transition system S_i used to compose $S_{\text{GlTime}}(\mathcal{N})$ $[S_{\text{LocTime}}(\mathcal{N})]$ (i.e. it is an assignment to the variables V_{S_i}). Note that a state s_i that corresponds to the encoding of the i-th automaton in $S_{\text{GlTime}}(\mathcal{N})$ and a state s'_i that corresponds to the encoding of the i-th automaton in $S_{\text{LocTime}}(\mathcal{N})$ are defined over the same set of variables, except for the time variable t_i. Thus, given a time variable t_i, we denote with $s'_i = \langle s_i, t \rangle$ the assignment equal to s_i and with the additional variable t_i.

Theorem 7 (Equivalence of thw two semantics [BJLY98]) *A state $s = \langle s_1, \ldots, s_n \rangle$ is reachable in $S_{\text{GlTime}}(\mathcal{N})$ iff there exists a synchronized state $s' = \langle \langle s_1, t_1 \rangle, \ldots, \langle s_n, t_n \rangle \rangle$ reachable in $S_{\text{LocTime}}(\mathcal{N})$.*

In the following, we prove a stronger version of the Theorem 7, which limits itself to the preservation of the reachability passing from local time to global time and vice versa. Here, we define a mapping between the paths of the local time and the paths of the global time encoding which preserves their abstract-time version.

Definition 15 (Abstract-time trace for $S_{\text{GlTime}}(\mathcal{N})$) *Given the global-time encoding $S_{\text{GlTime}}(\mathcal{N})$, we denote with $\mathcal{W}' = \mathcal{W} \setminus \{\delta_1, \ldots, \delta_n\}$ the set of input variables $\mathcal{W}$ of $S_{\text{GlTime}}(\mathcal{N})$ without the δ_i variables of each transition* *system S_i. Given a trace $w = w_1; \ldots ; w_k$, where w_i is an assignment to the input variables $\mathcal{W}$, its* time-abstract trace *$\alpha(w)$ is the trace obtained removing all the assignments w_i such that $w_i(\varepsilon) = \textsc{t}$ and restricting all the other assignments in w to assignments over $\mathcal{W}'$.*

In practice a time-abstract trace of the global time semantic is a trace that does not have timed events and does not have any assignments to the variables δ.

Definition 16 (Abstract-time trace for $S_{\mathbf{LocTime}}(\mathcal{N})$) *Given a trace of $S_{\text{LocTime}}(\mathcal{N})$ $w = w_1; \ldots ; w_k$, its* time-abstract trace *$\alpha(w)$ is the trace obtained removing all the assignments w_i such that such that $w_i(\varepsilon) = \textsc{t}$, for any $1 \leq j \leq n$.*

Theorem 8 *A state $s = \langle s_1, \ldots , s_n\rangle$ is reachable in $S_{\text{GlTime}}(\mathcal{N})$ with a path π over the time-abstract trace w iff there exists a synchronized state $s' = \langle\langle s_1, t_1\rangle, \ldots , \langle s_n, t_n\rangle\rangle$ reachable in $S_{\text{LocTime}}(\mathcal{N})$ with a path π' over w.*

Proof. ($\Leftarrow$) We define a function ϱ that maps a path of $S_{\text{GlTime}}(\mathcal{N})$ to a path of $S_{\text{LocTime}}(\mathcal{N})$. ϱ is defined by simply adding the time elapsed from the beginning of the path to the states and the shared events and by replacing a global timed transition with a sequence of equivalent local timed transitions. Formally, we define ϱ recursively on the length of the path. The definition keeps invariant that the last state of $\varrho(\pi)$ is synchronized. If π is a path of $S_{\text{GlTime}}(\mathcal{N})$, then $\varrho(\pi)$ is defined recursively as follows:

- if $k = 0$ and $\pi = \langle s_1, \ldots , s_n\rangle$, then $\varrho(\pi) := \langle\langle s_1, t_1 = 0\rangle, \ldots , \langle s_n, t_n = 0\rangle\rangle$; note that the last state of $\varrho(\pi)$ is synchronized;
- if $\pi = \pi'; w_k; \langle s_1, \ldots , s_n\rangle$, then
  - if w_k is a discrete event (i.e. $w_k(\varepsilon_i) \neq \textsc{t}$), then let t be the time of the last state of $\varrho(\pi')$ (which is synchronized by construction);

$\varrho(\pi) := \varrho(\pi'); w_{k_{|\mathcal{W}'}}; \langle\langle s_1, t_1 = t\rangle, \ldots, \langle s_n, t_n = t\rangle\rangle$; thus, the last state of $\varrho(\pi)$ is synchronized;

– if w_k is a timed event, then let t be the time of the last state of $\varrho(\pi')$ (which is synchronized by construction); $\varrho(\pi) := \varrho(\pi'); \langle\varepsilon_1 = \text{T}, \varepsilon_2 = \text{S}, \ldots, \varepsilon_n = \text{S}; \ldots; \langle\varepsilon_1 = \text{S}, \varepsilon_2 = \text{S}, \ldots, \varepsilon_n = \text{T}\rangle; \langle\langle s_1, t_1 = t\rangle, \ldots, \langle s_n, t_n = t\rangle\rangle$ (this is possible since the sets V_i are disjoint and the continuous evolution of the components are independent); thus, the last state of $\varrho(\pi)$ is synchronized.

We can easily prove by induction that the time-abstract trace accepted by π is the same time-abstract trace accepted by $\varrho(\pi)$.

($\Rightarrow$) Following the opposite direction, we define a function ϱ' that maps a path of $S_{\text{LocTime}}(\mathcal{N})$ that ends in a synchronized state into a path of $S_{\text{GlTime}}(\mathcal{N})$. ϱ' is defined by shuffling independent transitions so that all the components can take a timed transitions. Timed transitions are split to allow components to take a discrete transition when necessary. Formally, we define ϱ' recursively on the length of the path. We use the concept of first i-th step to denote the first transition labeled with an event of the i-th component. If π is a path that ends in a synchronized state of $S_{\text{LocTime}}(\mathcal{N})$, then $\varrho'(\pi)$ is defined recursively as follows:

- if $k = 0$ and $\pi = \langle\langle s_1, t_1\rangle, \ldots, \langle s_n, t_n\rangle\rangle$, then $\varrho'(\pi) := \pi = \langle s_1, \ldots, s_n\rangle$;

- if, for some i, the first i-th step is a discrete event $\varepsilon_i = a$, with $a \in A_i$. We consider the first of such i, and without changing the time-abstract trace, take such step as first; thus $\pi = \langle\langle s_1, t_1\rangle, \ldots, \langle s_n, t_n\rangle\rangle; w_k; \pi'$, with $w_k(\varepsilon_i) \in A_i$, then $\varrho'(\pi) := \langle s_1, \ldots, s_n\rangle; w_{k_{|\mathcal{W}}}; \varrho'(\pi')$;

- if, for all i, the first i-th step is a timed event, we consider the smallest delta time δ of such timed events, and without changing the time-abstract trace, we refine the path by splitting each of these timed transitions into two, of which the first takes δ time; still without changing the time-abstract trace, we take such timed steps as first; thus $\pi = \langle\langle s_1, t_1 = t\rangle, \ldots, \langle s_n, t_n = t\rangle\rangle; w_1; \ldots; w_i; \ldots; \pi'$, where $w_j(\varepsilon_j = \text{T})$ for $j = 1, \ldots, i$ and π' starts with $\langle\langle s_1, t_1 = t + \delta\rangle, \ldots, \langle s_n, t_n = t + \delta\rangle\rangle$, thus, $\varrho'(\pi) := \langle s_1, \ldots, s_n\rangle; ; \langle\varepsilon_1 = t, \delta_i = \delta\rangle, \ldots, \langle\varepsilon_n = t, \delta_n = \delta\rangle; \varrho'(\pi')$.

We can easily prove by induction that the time-abstract trace accepted by π is the same time-abstract trace accepted by $\varrho'(\pi)$. $\diamond$

Chapter 5

Time-Aware Relational Abstraction

Note. The material presented in this chapter has already been presented in [MCTT13].

This work was supported in part by NSF grants CSR-0917398 and SHF:CSR-1017483.

In the previous chapters we presented a precise encoding of different classes of hybrid systems in symbolic transition systems. The encoding enabled the use of different verification techniques (e.g. SMT-based verification algorithms) to prove safety properties of the original hybrid automaton.

However, the practical application of the approach may be limited for several reasons. First, the encoding requires that, for each flow condition, there exists an explicit solution expressible in a theory that can be effectively handled by modern SMT solvers. For example, the general solution of a linear system cannot be encoded in $\mathcal{T}(\mathbb{R})$. Second, even if an encoding in $\mathcal{T}(\mathbb{R})$ exists, the model checking procedures may not scale, due to the intrinsic complexity in the satisfiability problems for such theory. As the experimental results of Section 10.2.1 shows, the current performances of SMT solvers on non-linear real arithmetic formulas are still not satisfactory and their scalability is an issue, despite the recent improvements [JdM12].

One viable approach to tackle the problem consists to analyze an abstraction of the hybrid automaton that approximates its behaviors conservatively. *Relational abstraction* [ST11a] is an effective abstraction technique that enable SMT-based verification for *Linear Hybrid Systems*. Relational abstraction replaces the ordinary differential equations in each location of the hybrid automaton with a binary relation over the continuous state space of the system. The relation approximates the binary reachability relation induced by the ordinary differential equations. Thus, by replacing the ODEs in each mode by their relational abstraction, it is possible to over-approximate a linear hybrid system with an infinite-state transition system.

Relational abstraction has been applied successfully to verify several systems [ST11a, Tiw12, TD12]. The main computation technique of relational abstractions for linear hybrid systems is implemented in the the HybridSAL [Tiw12] tool. However, HybridSAL currently generates one fixed abstraction for any given ODE. Hence, if the relational abstraction fails to prove a valid property, then there is no option for refining the abstraction. Then, the abstractions generated by HybridSAL are time-agnostic: they do not relate the time elapsed in the continuous transitions with the state variables.

Time-agnostic relational abstractions can prove several nontrivial properties, but they may fail in the following cases:

1. Since the relational abstraction is time-agnostic, it cannot be used to verify timing properties.

2. The time-agnostic abstractions are very coarse (imprecise). In fact, they often do not capture the relationship between different variables of the system. Even in simple cases, the loss of information in time-agnostic relational abstraction is very significant.

3. Finally, the time-agnostic abstraction is not suitable to prove properties of distributed systems, like networks of hybrid automata. While the abstraction is computed compositionally for each location of each automaton, it looses the relationship between the variables of different components. This effect may be seen as a corollary of the previous issue.

In this chapter we present a technique that allow us to compute a more precise, *time-aware*, relational abstraction[1]. Time-aware relational abstraction overcomes the shortcomings of relational abstraction procedure implemented in HybridSAL. First, it is not time-agnostic and it contains information about the relationship between the state variables and the time-elapse variable. Second, time-aware relational abstraction enables tuning the precision of the abstraction. Third, the relation over time implicitly relates the variables in the different automata in the network, thus enabling the analysis of distributed systems.

Our main contribution is a procedure for computing a time-aware relational abstraction that can be tuned to achieve any desired precision-efficiency trade-off. The abstraction can be made more and more precise, but that increases the cost of analyzing it using SMT-based techniques (e.g. bounded model checking, k-induction, IC3).

Our approach for generating time-aware relational abstractions is based on exploiting the eigenstructure of the matrix A of the linear ordinary differential equations. The technique for creating time-agnostic relational abstractions, as implemented in HybridSAL [Tiw12], was also based on the same basic idea. However, we have to non-trivially extend that approach to preserve information about time (rather than throwing it away) in the abstraction. The key challenge in extending the procedure for creating

[1]We consider *autonomous systems*, where the vector field of the system (i.e. the right-hand side of the ODEs) does not explicitly depends on time.

time-agnostic relational abstractions to time-aware relational abstractions is that the relationship between the time elapsed (Δt) and the change in the value of any state variable (Δx) is seldom linear. It is often nonlinear, and it often contains quadratic terms and transcendental functions, such as the exponential function and the trigonometric functions. Since we wish to effectively use SMT verification techniques, we need the abstract system to be encoded in "easy" theories, namely, in linear rational arithmetic ($\mathcal{T}(\mathbb{Q})$). The main technical contribution is the creation of a time-aware relational abstraction of linear ODEs using piecewise linear approximations of nonlinear (transcendental) functions.

In the experimental evaluation of Section 10.2.2 we show the effectiveness of our approach on two case studies, a PID controller and an active suspension controller. We show that, while the previous relational abstraction was too coarse, time-aware abstraction is able to verify time properties on both benchmarks.

5.1 Relational Abstraction

In the following, we consider a *Linear Hybrid System* $H = \langle V, X, Init, Invar, Trans, Flow \rangle$ such that, for each explicit location $q \in Q$, the flow condition is a linear ODE[2]:

$$Flow(q) = A\vec{x} + \vec{b}$$

where $A \in \mathbb{R}^{n \times n}$, $b : \mathbb{R}^n$, $\vec{x}$ is the vector of X and $\dot{\vec{x}}$ is the vector of $\dot{X}$, n is the dimension of the dynamical system (i.e. the cardinality of the set X, $|X| = n$) and $\vec{x}$ is a vector of all the continuous variables X.

Definition 17 *An infinite-state transition system S is an* abstraction *of the hybrid system H if for all traces $\pi_H = \langle l_0, v_0 \rangle \xrightarrow{\delta_1} \dots \xrightarrow{\delta_k} \langle l_k, v_k \rangle$ of H*

[2]We deal with *autonomous systems* where the vector of inputs $\vec{b}$ does not depend on the time variable.

there exists a path $s_0; \delta_1; s_1; \delta_2; \ldots; \delta_k; s_k$ *of* S.

Relational abstraction [ST11a] abstracts only the continuous evolution in each location of the hybrid automaton, leaving unchanged its discrete locations and transitions.

Thus, the relational abstraction of H is the transition system $S_H = \langle V_S, \mathcal{W}_S, Init_S, Inv_S, Trans_S \rangle$:

- $V_S := V \cup X \cup \{t\}$,
- $\mathcal{W}_S := \{\}$,
- $Init_S := t = 0 \wedge Init_H$,
- $Inv_S := Invar_H$
- $Trans_S :=$ UNTIMED $\vee$ TIMED where
  - UNTIMED $:= t' = t \wedge Trans_H(V, X, V', X')$.
  - TIMED $:= t' > t \wedge \bigwedge_{q \in Q} \textsc{Timed}_q(X, X')$

The timed transition TIMED abstracts the continuous evolution in each location. In particular, $\textsc{Timed}_q$ relates all the values assigned to the continuous variables with all the possible future values assigned to the continuous variables after a continuous transition.

Definition 18 (Relational Abstraction) $\textsc{Timed}_q$ *is a* relational abstraction *for the location* $q \in Q$ *if* $\forall v_i \in \mathbb{R}^n, \delta \in \mathbb{R}$, *it is the case that:*

$$\langle q, X = v_i, q', X' = f_q(v_i, \delta) \rangle \models \textsc{Timed}_q$$

where $f_q : \mathbb{R}^{n+1} \to \mathbb{R}^n$ *is the solution to the system of ODEs* $Flow(q)$.

It follows immediately from the definitions above that the infinite-state transition system S_H is an abstraction of H.

5.1.1 Eigenstructure-based relational abstraction

There exists several instantiation of relational abstractions [ST11a, Tiw12, ZST12, DM12]. Their strengths and disadvantages are discussed in the related work Section of this chapter (Section 5.4).

In the following, we describe the relational abstraction technique for linear systems presented in [Tiw12] , that we will extend in the next section to increase its precision.

The relational abstraction $\textsc{Timed}_q$ of $Flow(q)$ may be computed exploiting the eigenstructure of the matrix A (i.e. all the pairs of eigenvalues and eigenvectors of A). First, the system of differential equations is rearranged partitioning the variables $\vec{x}$ into two disjoint vectors $\vec{y}$ and $\vec{z}$ such that[3]:

$$\begin{bmatrix} \dot{\vec{y}} \\ \dot{\vec{z}} \end{bmatrix} = \begin{bmatrix} A_1 & A_2 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \vec{y} \\ \vec{z} \end{bmatrix} + \begin{bmatrix} \vec{b_1} \\ \vec{b_2} \end{bmatrix}$$

Given an $n \times n$ matrix M, let Λ_M be the set of all the pairs $\langle \lambda, \vec{c} \rangle$, where λ is a left-eigenvalue of the matrix M and $\vec{c}$ is one of its associated left-eigenvectors. More precisely, for each eigenvalue λ we consider only linearly independent eigenvectors. Let R be a set of tuples $\langle r, \vec{c} \rangle$ such that $\frac{d\vec{c}^T\vec{x}}{dt} = r$, for some $r \in \mathbb{R}$. Note that, for each $z_i \in \vec{z}$, we can easily get one of such tuple $\langle b_i, z_i \rangle$, since $\frac{dz_i}{dt} = b_i$.

The abstraction $\textsc{Timed}_q$ is computed from each pair $\langle \lambda, \vec{c} \rangle \in \Lambda_{A_1}$:

$$\textsc{Timed}_q := \bigwedge_{\langle \lambda, \vec{c} \rangle \in \Lambda_{A_1}} \phi_{\langle \lambda, \vec{c} \rangle} \wedge \bigwedge_{\substack{\langle r, \vec{c} \rangle, \langle l, \vec{d} \rangle \in R \\ r \neq l \vee \vec{c} \neq \vec{d}}} \frac{\vec{c'} - \vec{c}}{r} = \frac{\vec{d'} - \vec{d}}{l}$$

where $\phi_{\langle \lambda, \vec{c} \rangle}$ is a formula that depends on the kind of eigenvalues and eigenvectors, which may be real or complex.

[3]Note that if there are no variables with constant derivative the dimension of $\vec{z}$ is 0, which is a simpler special case.

If $\vec{c} \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}$, the abstraction is defined with the predicate $p(\vec{x}) = \vec{c}^T\vec{y} + \vec{d}^T\vec{z} + e$, where $\vec{d}^T = \vec{c}^T\frac{A_2}{\lambda}$, $e = \frac{\vec{c}^T\vec{b_1} + \vec{d}^T\vec{b_2}}{\lambda}$. The details [Tiw12] about the computation of $p(\vec{x})$ are reported in the Appendix A.2.

In the following, we use p to denote the linear expression $p(\vec{x})$ and p' to denote the linear expression $p(\vec{x}')$ over the next-state variables.

$$\phi_{\langle\lambda,\vec{c}\rangle} := \begin{cases} (p' \leq p < 0) \vee (0 < p \leq p') \vee (0 = p = p') & \text{if } \lambda > 0 \\ (p \leq p' < 0) \vee (0 < p' \leq p) \vee (0 = p = p') & \text{if } \lambda < 0 \\ t' - t = \frac{\vec{c}^T(\vec{y'} - \vec{y})}{\vec{c}^T\vec{b_1}} & \lambda = 0 \end{cases}$$

If $\vec{c} \in \mathbb{C}^n$ and $\lambda \in \mathbb{C}$, the abstraction is defined using two predicates, $p_1(\vec{x})$ and $p_2(\vec{x})$. Suppose $\vec{c} = \vec{d} + i\vec{e}$ and $\lambda = \alpha + i\beta$. We define p_1 and p_2 as follows [4]: $p_1(\vec{x}) = \vec{d}^T\vec{y} + \vec{c_1}^T\vec{z} + e_1$, $p_2(\vec{x}) = \vec{e}^T\vec{y} + \vec{c_2}^T\vec{z} + e_2$ and $c_1, c_2, \vec{c_1}^T, \vec{c_2}^T, e_1, e_2$ are such that $\dot{p_1} = \alpha p_1 - \beta p_2$ and $\dot{p_2} = \beta p_1 + \alpha p_2$. The formula $\phi_{\langle\lambda,\vec{c}\rangle}$ added to $\textsc{Timed}_q$ is:

$$\phi_{\langle\lambda,\vec{c}\rangle} := \begin{cases} p_1^2(\vec{x}) + p_2^2(\vec{x}) \geq p_1^2(\vec{x'}) + p_2^2(\vec{x'}) & \text{if } \alpha \leq 0 \\ p_1^2(\vec{x'}) + p_2(\vec{x'}) \geq p_1^2(\vec{x}) + p_2^2(\vec{x}) & \text{if } \alpha \geq 0 \end{cases}$$

A practical requirement for the abstraction is that it has to be expressed in a formula in the *Linear Real Arithmetic* Theory. Note that the abstraction, $\phi_{\langle\lambda,\vec{c}\rangle}$, generated in the complex eigenvalue case is non-linear and thus it is approximated in the original approach.

5.2 Simple Motivating Example

We illustrate the main idea underlying time-aware relational abstraction with a simple contrived example.

[4]See the Appendix A.2 for details

Consider the linear system

$$\begin{aligned}\dot{x} &= -2x,\\ \dot{y} &= 0.5 - y\end{aligned}$$

with initial condition $x \in [-1, 0.9], y \in [1.1, \infty)$. We want to prove that x is always less-than y; that is, $\mathtt{G}(x < y)$.

One way to prove safety of such systems involves constructing a relational abstraction of the system and then verifying the abstract system using infinite bounded model checking and k-induction.

Relational abstraction replaces the differential equation by a discrete transition. The abstract transition relates the current value of x, y with *any future* value x', y'. The default relational abstraction, constructed by HybridSAL, for the above differential equation is the following transition:

$$\begin{aligned}(x = x' = 0 \;\vee\; 0 < x' \leq x \;\vee\; 0 > x' \geq x) \;\;\wedge\\ (y = y' = 0.5 \;\vee\; 0.5 < y' \leq y \;\vee\; 0.5 > y' \geq y)\end{aligned}$$

In the abstract system, there is a transition from (x, y) to a new state (x', y') if these four values satisfy the above constraint.

The abstraction is sound: starting from (x, y) and following the solutions of the differential equations, if we reach (x', y') at any future time instance, then x, y, x', y' will necessarily satisfy the above constraint. However, the relational abstraction is very coarse (imprecise). In particular, the rate at which x and y are changing is abstracted away. If $\dot{x} = -2x$ were replaced by $\dot{x} = -0.2x$, we would still get the same relational abstraction.

As a result of this imprecision, the default relational abstraction is insufficient to prove the safety property. (We get a spurious counterexample when trying to prove the safety property using the above relational abstraction.)

We construct more precise relational abstractions, called *time-aware* relational abstraction. First, we make the implicit time variable explicit by adding a new variable called $\mathtt{t}$ to the state space and the differential equation $\dot{\mathtt{t}} = 1$ to the dynamics. Time t is the glue that will help relate the change in x to the change in y.

The new time-aware relational abstraction relates the old values x, y, t to the new values x', y', t'. For the above system, the new abstract transition is the conjunction of two constraints: the first constraint, shown below, relates x, x', t, t':

$$\begin{aligned}
&(x = x' = 0) \;\vee \\
&(0 < x' \leq x \;\wedge\; \ln_{\mathtt{lb}}(x) - \ln_{\mathtt{ub}}(x') \geq -2(t' - t) \geq \\
&\qquad\qquad\qquad \ln_{\mathtt{ub}}(x) - \ln_{\mathtt{lb}}(x')) \;\vee \\
&(0 > x' \geq x \;\wedge\; \ln_{\mathtt{lb}}(-x) - \ln_{\mathtt{ub}}(-x') \geq -2(t' - t) \geq \\
&\qquad\qquad\qquad \ln_{\mathtt{ub}}(-x) - \ln_{\mathtt{lb}}(-x'))
\end{aligned}$$

There is a similar second constraint that relates y, y', t, t'.

There are two key points to note here. First, there is no simple *linear* relationship between the time variable and the x, y variables. They are related through the natural logarithm (ln) function; specifically, $x(t) = x(0)e^{-2t}$ is (part of) the solution of the above differential equation; and hence we have $-2(t' - t) = \ln(x') - \ln(x)$. To enable analysis using BMC/SMT tools, we need a *piecewise linear* function that computes a sound lower- and upper-bound of the ln function. These approximate functions, called $\ln_{\mathtt{lb}}$ and $\ln_{\mathtt{ub}}$ respectively, will be defined later.

Second, all interactions between different state variables are captured via the time variable. Note that the first constraint above relates x, x' with the time elapsed $t' - t$, and similarly the second constraint relates y, y' with the time elapsed $t' - t$. By reasoning over the conjunction, we deduce the relationship between x and y.

In particular, using the refined time-aware relational abstraction, we can prove the safety property $\mathtt{G}(x < y)$.

5.3 Time-Aware Relational Abstraction

A time-aware relational abstraction of a dynamical system is a binary relation that holds between the current state of the system, including the current time, and any future reachable state of the system (including the future time). In this section, we describe a procedure for constructing time-aware relational abstractions of linear dynamical systems; that is, systems whose dynamics are specified using linear ordinary differential equations of the form $\dot{\vec{x}} = A\vec{x} + \vec{b}$.

5.3.1 Overall Approach

Consider a linear system $\dot{\vec{x}} = A\vec{x} + \vec{b}$. The exact relationship between $t' - t$ and the variables $\vec{x}$ is given by the explicit solution:

$$\vec{x}(t) \ := \ x_0 e^{At} + \int_{s=0}^{t} e^{(t-s)A}\vec{b}\, \mathrm{d}s \tag{5.1}$$

where x_0 is the state of the dynamical system. It is hard to reason with this solution directly. In some very special cases, this explicit solution can be used to effectively solve the reachability problem for linear systems [LPY01]. This happens, for example, when either A is nilpotent, or its eigenvalues are either all reals or all purely imaginary, but even in these restricted cases, the solution requires reasoning over nonlinear real arithmetic. Hence, working with the explicit solution in Equation 5.1 is not very practical.

In our approach, we create an abstraction of the solution. Specifically, we construct a relationship between the current value $\vec{x}$ of the state variables, the future value $\vec{x}'$ of the state variables, the current value t of time, and the future value t' of time. Note that we use the shorthand $\delta = t' - t$ to represent the relationship of t before and after the continuous transition.

The relationship we construct will over-approximate the binary reachability relation.

To ease the presentation, we assume that the set of continuous variables X of the hybrid automaton H contains a clock variable t, which counts the total time elapsed in the system. Initially t is 0 and its derivative is 1 in all the locations. Also, t is never used in a jump or in an invariant condition[5]. Note that we defined the clock variable t in the transition system encoding S_H of H. In the following, suppose that the transition system encoding S_H of H does not define an additional variable t, but it uses the continuous variable t already present in H.

In the following, let R be a set of tuples $\langle r, \vec{c}\rangle$ such that $\frac{d\vec{c}^T\vec{x}}{dt} = r$, for some $r \in \mathbb{R}$. For each location $q \in Q$, for each $\langle \lambda, \vec{c}\rangle \in \Lambda_{A_1}$ and for each $\langle r, \vec{c}\rangle \in R$, we obtain the following time-aware relational abstraction:

$$\textsc{Timed}^t_q \;:=\; \bigwedge_{\langle\lambda,\vec{c}\rangle\in\Lambda} \phi^t_{\langle\lambda,\vec{c}\rangle} \;\wedge\; \bigwedge_{\langle r,\vec{c}\rangle\in R} \phi^c_{\langle r,\vec{c}\rangle} \tag{5.2}$$

where $\phi^t_{\langle\lambda,\vec{c}\rangle}$ is the time-aware abstraction generated from $\langle\lambda,\vec{c}\rangle$ and $\phi^c_{\langle r,\vec{c}\rangle}$ is the abstraction generated for linear expressions with a constant-rate derivative. We define $\phi^t_{\langle\lambda,\vec{c}\rangle}$ and $\phi^c_{\langle r,\vec{c}\rangle}$ below. As in the case of the eigenstructure-based abstraction, the definition of $\phi^t_{\langle\lambda,\vec{c}\rangle}$ depends on whether the eigenvalue λ is real or complex.

In the following, we use p to denote the linear expression $p(\vec{x})$, and p' to denote the linear expression $p(\vec{x}')$ over the next-state variables.

5.3.2 Constant Rate

We define $\phi^c_{\langle r,\vec{c}\rangle}$ now. Consider the linear expression $p = \vec{c}^T\vec{x}$ such that $\dot{p} = r$, where r is a nonzero constant, then we get the following time-aware

[5]If we are not interested in the total amount of time elapsed, we can keep only the time elapsed during a continuous transition, δ, as shown in Remark 2.

relational abstraction $\phi^c_{\langle r,\vec{c}\rangle}$:

$$\phi^c_{\langle r,\vec{c}\rangle} \;:=\; (p' - p) \;=\; r\delta \tag{5.3}$$

where t is the time variable. We can then add the conjunct $\phi^c_{\langle r,\vec{c}\rangle}$ to the time-aware relational abstraction of linear system.

Note that we can obtain a predicate $\langle b_{2_i}, z_i\rangle$ of R from each variable z_i of $\vec{z}$. In practice, the variables in $\vec{z}$ have a constant derivative, and thus the encoding of their dynamic is exact and similar to the one shown in Example 2.

5.3.3 Real Eigenvalues

To ease the presentation of the real and complex eigenvalues cases, we consider a linear system where $\vec{b} = \vec{0}$. This means that we deal with a continuous dynamic of the form $\dot{\vec{x}} = A\vec{x}$. The terms $\vec{b}$ complicates the computation of the linear constraints that will be used in the abstraction. The computation of these constraints in the case $\vec{b} \neq \vec{0}$ has been already defined in [Tiw12], and it is fully reported in the Appendix A.2. Note that the results regarding time-aware relational abstraction extends smoothly in the case $\vec{b} \neq \vec{0}$: in fact, once the additional predicates have been computed from $\dot{\vec{x}} = A\vec{x} + \vec{b}$, the steps needed to compute the abstraction do not change.

In this section we define $\phi^t_{\langle\lambda,\vec{c}\rangle}$ in the case λ is real. Let $\vec{c}$ be a left eigenvector of A corresponding to some real eigenvalue λ; that is,

$$\vec{c}^T A = \lambda\vec{c}^T$$

where $\vec{c}^T$ is a row vector obtained by transposing (the column vector) $\vec{c}$. (Equivalently, $\vec{c}$ is a eigenvector of the transpose of A). Consider the linear expression

$$p(\vec{x}) \;=\; \vec{c}^T\vec{x}$$

Clearly, we have

$$\frac{dp(\vec{x})}{dt} = \frac{d\vec{c}^T\vec{x}}{dt} = \vec{c}^T\frac{d\vec{x}}{dt} = \vec{c}^T A\vec{x} = \lambda\vec{c}^T\vec{x} = \lambda p(\vec{x})$$

Hence, the value of p changes exponentially; that is:

$$p(\vec{x}(t)) = p(\vec{x}(0))e^{\lambda t} \tag{5.4}$$

The linear expression p can be used to constrain the future value, $\vec{x}'$, and the current value, $\vec{x}$, of the state variables. When $\lambda > 0$, the following constraint holds between any future value $\vec{x}'$ and the current value $\vec{x}$ of the state variables:

$$\begin{aligned}\psi(p,p') := (p = p' = 0) \quad \vee \\ &(0 < p \leq p' \quad \wedge \quad \ln_{\texttt{lb}}(p') - \ln_{\texttt{ub}}(p) \leq \lambda\delta \leq \ln_{\texttt{ub}}(p') - \ln_{\texttt{lb}}(p)) \quad \vee \\ &(0 > p \geq p' \quad \wedge \quad \ln_{\texttt{lb}}(-p') - \ln_{\texttt{ub}}(-p) \leq \lambda\delta \leq \ln_{\texttt{ub}}(-p') - \ln_{\texttt{lb}}(-p))\end{aligned}$$

When $\lambda < 0$, the constraint is the same as above, but with p and p' swapped. We thus have

$$\phi^t_{\langle\lambda,\vec{c}\rangle} = \begin{cases} \psi(p,p') & \text{if } \lambda > 0 \\ \psi(p',p) & \text{if } \lambda < 0 \\ p = p' & \text{if } \lambda = 0 \end{cases} \tag{5.5}$$

where ψ is as defined above.

We remark that, in practice, we only need a linear expression $p(\vec{x})$ that satisfies the equation $\frac{dp(\vec{x}(t))}{dt} = \lambda p$. For linear dynamical systems, such a p can be found using the left eigenvectors of the A matrix. For nonlinear dynamics, one needs to develop other techniques to obtain such a p, but once found, it can be used to construct time-aware relational abstractions of nonlinear systems too [TK04].

We now define the functions $\ln_{\texttt{lb}}$ and $\ln_{\texttt{ub}}$. These functions are piecewise linear approximations of the (lower and upper bounds for the) nonlinear natural logarithm function ln; that is, they satisfy the following condition:

$$\ln_{\texttt{lb}}(x) \;\leq\; \ln(x) \;\leq\; \ln_{\texttt{ub}}(x), \quad \forall x \in \mathbb{R}^+$$

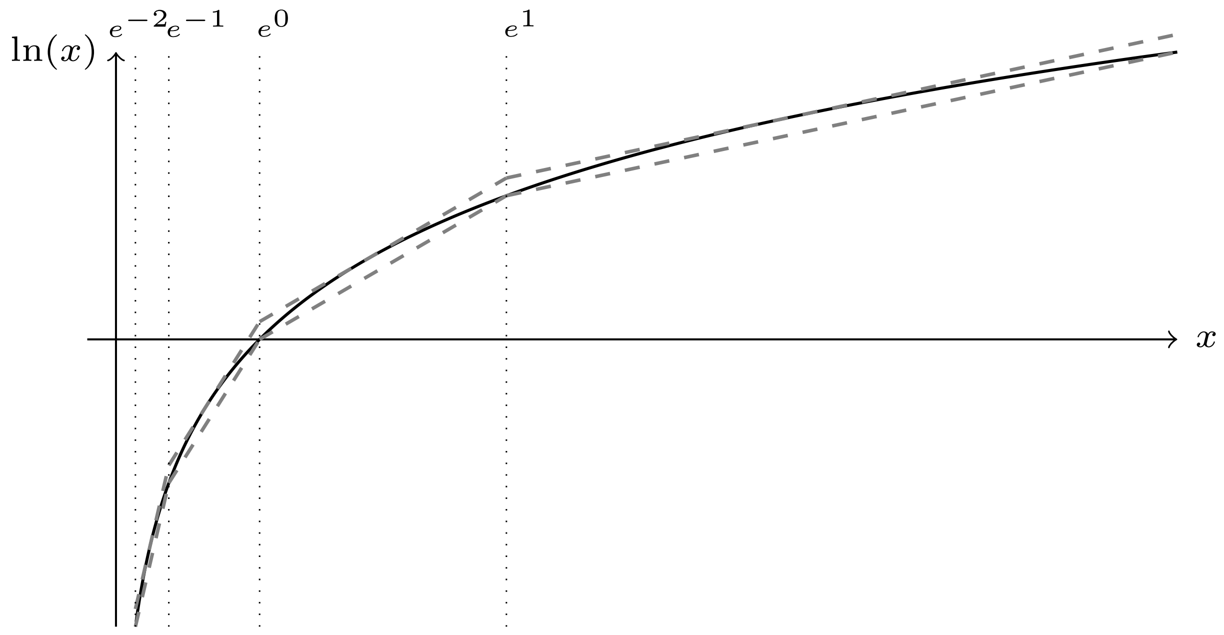


Figure 5.1: Piecewise linear approximation for natural logarithm function. The solid line plots $\ln(x)$ and the dotted lines shows the piecewise linear under- and over-approximations.

Piecewise Linear Approximation for Natural Logarithm

The natural logarithm function, $\ln(x)$, can be approximated using a piecewise linear function, as described in [Hil00]. Figure 5.1 illustrates this approximation.

We first divide the real number line into infinitely many intervals. Consider the infinitely many intervals $I_k := [e^k, e^{k+1}]$, $k \in \mathbb{Z}$. Clearly, we have $\bigcup_{k \in \mathbb{Z}} I_k = (0, \inf)$.

Since the logarithm function is concave, it is easy to obtain a piecewise linear underapproximation of the function. This underapproximation is obtained by just linearly extrapolating within each interval, as shown in Figure 5.1. Specifically, if x is in the interval I_k, then $\ln(x)$ is approximated by:

$$\ln_{\texttt{lb}}(x) \;=\; \frac{e^{-k}}{e-1}x + k - \frac{1}{e-1} \qquad (5.6)$$

This idea for approximation works not only for the base e, but also for any base $a > 1$. For base e, the approximation error is defined by $\gamma(x) = \ln(x) - \ln_{\mathtt{lb}} x$. In any interval I_k, γ is bounded by

$$\ln(e-1) - 1 + \frac{1}{e-1} \tag{5.7}$$

Thus, in general, γ depends only on the base of the logarithm and not on the interval itself.

The function $\ln_{\mathtt{lb}}(x)$ gives a lower bound for the function $\ln(x)$. The upper bound can be obtained by just adding γ to the lower bound.

$$\ln_{\mathtt{ub}}(x) \;=\; \frac{e^{-k}}{e-1}x + k - 1 + \ln(e-1) \tag{5.8}$$

Thus, we get a piecewise linear function for both under and over approximating the natural logarithm function.

In practice, we cannot use the above piecewise linear function because it is defined over infinitely many intervals. We pick finitely many intervals.

In our implementation, we use two parameters l, m to specify the intervals that are used to create the piecewise linear lower- and upper-bound functions. Given natural numbers l, m, our implementation uses the intervals

$$(-\infty, e^{-l}],\ [e^{-l}, e^{-l+1}],\ \ldots,\ [e^{m-1}, e^{m}],\ [e^{m}, \infty) \tag{5.9}$$

We use linear-interpolation based approximation on the bounded intervals and we use a sound coarse approximation on the unbounded intervals. Clearly, we can *refine* our abstraction by increasing the number of intervals; that is, by increasing the values of the parameters l, m. Automated abstraction-refinement of this kind is left for future work.

One advantage of this approximation is that the size of the intervals grows exponentially. Hence, a few intervals can approximate the logarithm for a “reasonable” range of x values. Also, note that the error is bounded and depends only on the base of the logarithm. Thus, changing the base of the logarithm provides another way to get a better approximations (refinements).

5.3.4 Complex Eigenvalues

We define $\phi^t_{\langle\lambda,\vec{c}\rangle}$ for $\lambda \in \mathbb{C}$. Consider a linear dynamical system $\dot{\vec{x}} = A\vec{x}$, where A contains only real (in practice, rational) entries. Let $\vec{c} := \vec{d} + \iota\vec{e}$ be a left eigenvector of A corresponding to the complex eigenvalue $\lambda := a + \iota b$; that is,

$$(\vec{d}^T + \iota\vec{e}^T)A \;=\; (a + \iota b)(\vec{d}^T + \iota\vec{e}^T)$$

Now, consider the two linear expressions

$$p(\vec{x}) \;=\; \vec{d}^T\vec{x} \qquad q(\vec{x}) \;=\; \vec{e}^T\vec{x}$$

Computing the time derivative (Lie derivative) of these two expressions, we find that the expressions p and q satisfy the differential equation $\dot{p} = ap - bq, \dot{q} = bp + aq$, and hence the closed-form solution for them is given by

$$\begin{aligned} p(\vec{x}(t)) &= re^{at}\cos(bt + \phi) \\ q(\vec{x}(t)) &= re^{at}\sin(bt + \phi) \end{aligned}$$

where r, ϕ are determined by the initial conditions (that is, values of $p(\vec{x}(0))$ and $q(\vec{x}(0))$) as follows:

$$\begin{aligned} r^2 &= p(\vec{x}(0))^2 + q(\vec{x}(0))^2 \\ \tan(\phi) &= q(\vec{x}(0))/p(\vec{x}(0)) \end{aligned}$$

We want to find linear relationships that hold between the initial value p, q of these two expressions, any future value p', q' of these two expressions, and the initial value t of the time, and the future value t' of time. We divide the task into two parts: first, we will find relationships $\phi^{\mathtt{ampl}}_{\langle\lambda,\vec{c}\rangle}$ that result from the exponential change in the amplitude (re^{at}) with time, and second, we will find relationships $\phi^{\mathtt{phase}}_{\langle\lambda,\vec{c}\rangle}$ that result from the linear change in phase ($bt + \phi$) with time. Then, the conjunct added to the time-aware relational abstraction will be

$$\phi^{t}_{\langle\lambda,\vec{c}\rangle} \ := \ \phi^{\mathtt{ampl}}_{\langle\lambda,\vec{c}\rangle} \wedge \phi^{\mathtt{phase}}_{\langle\lambda,\vec{c}\rangle} \tag{5.10}$$

where $\phi^{\mathtt{ampl}}_{\langle\lambda,\vec{c}\rangle}$ and $\phi^{\mathtt{phase}}_{\langle\lambda,\vec{c}\rangle}$ will be defined below.

Relating Amplitude and Time

Let us denote $p(\vec{x}(t'))$ by p' and $p(\vec{x}(t))$ by p; and similarly for q, q'. We are given p, q and the fact that

$$(p'^2 + q'^2)^{0.5} = (p^2 + q^2)^{0.5} e^{a\delta} \tag{5.11}$$

We wish to find linear constraints that are implied by the above equation. Those linear constraints can then be added to the time-aware relational abstraction without compromising soundness.

We use the piecewise linear approximation of the natural logarithm to deal with the exponential in the above expression, but there still remains the problem of handling the other quadratic sub-expressions. We will use coarse linear lower- and upper-bounds for the quadratic sub-expressions to finally obtain the conservative linear constraint that approximates Equation 5.11. Specifically, the following derivation shows how we obtain the linear approximation $\phi^{\mathtt{ampl}}_{\langle\lambda,\vec{c}\rangle}$ of Equation 5.11:

$$\begin{array}{rrcl} & (p'^2+q'^2)^{0.5} & = & (p^2+q^2)^{0.5}e^{a\delta} \\ \Rightarrow & a\delta & = & \ln(p'^2+q'^2)^{0.5} - \ln(p^2+q^2)^{0.5} \\ \Rightarrow & a\delta & \leq & \ln_{\mathtt{ub}}(\mathtt{q}_{\mathtt{ub}}(p'^2+q'^2)^{0.5}) - \\ & & & \ln_{\mathtt{lb}}(\mathtt{q}_{\mathtt{lb}}(p^2+q^2)^{0.5}) \quad \wedge \\ & a\delta & \geq & \ln_{\mathtt{lb}}(\mathtt{q}_{\mathtt{lb}}(p'^2+q'^2)^{0.5}) - \\ & & & \ln_{\mathtt{ub}}(\mathtt{q}_{\mathtt{ub}}(p^2+q^2)^{0.5}) \end{array}$$

Let $\phi^{\mathtt{ampl}}_{\langle\lambda,\vec{c}\rangle}$ denote the last conjunction above.

We have already defined the functions $\ln_{\mathtt{lb}}$ and $\ln_{\mathtt{ub}}$ before. We now define the functions $\mathtt{q}_{\mathtt{lb}}$ and $\mathtt{q}_{\mathtt{ub}}$ that compute linear lower and upper bound for the expression $(x^2+y^2)^{0.5}$. In other words, $\mathtt{q}_{\mathtt{lb}}$ and $\mathtt{q}_{\mathtt{ub}}$ are piecewise linear and satisfy the following condition:

$$\mathtt{q}_{\mathtt{lb}}((x^2+y^2)^{0.5}) \ \leq \ (x^2+y^2)^{0.5} \ \leq \ \mathtt{q}_{\mathtt{ub}}((x^2+y^2)^{0.5})$$

We use the value $|x|+|y|$ as a linear upper bound for $(x^2+y^2)^{0.5}$ and the expression $\mathtt{max}(|x|,|y|)$ as the lower bound for $(x^2+y^2)^{0.5}$.

$$\begin{array}{rcl} \mathtt{q}_{\mathtt{lb}}((x^2+y^2)^{0.5}) & := & \mathtt{max}(|x|,|y|) \\ \mathtt{q}_{\mathtt{ub}}((x^2+y^2)^{0.5}) & := & |x|+|y| \end{array}$$

Note that the functions $\mathtt{q}_{\mathtt{lb}}$ and $\mathtt{q}_{\mathtt{ub}}$ are both piecewise linear functions.

We thus get a linear and sound relationship between the current values p, q, the future values p', q' of the two linear expressions and the current and future values t, t' of time based on analyzing the change in the amplitude with time.

Relating Phase and Time

We now consider the problem of finding a linear relationship between p, p', q, q', t, t' based on analyzing the change in the phase with time. We

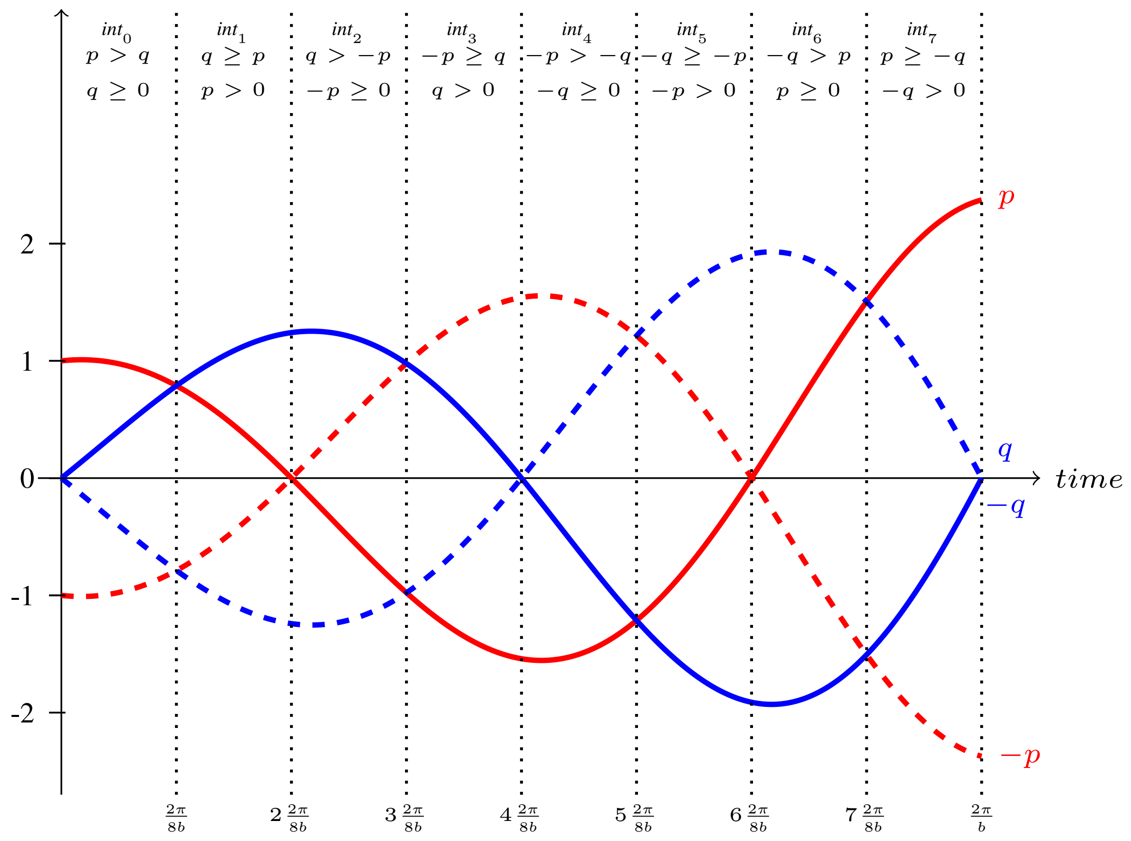


Figure 5.2: Extracting time elapsed information from analyzing the phase of the sinusoidal signals. The red line shows p, the blue line shows q, and the partition of the time axes based on the sign of p, q and $p \geq q$.

are given p, q and the fact that

$$\begin{aligned} p' &= (p^2+q^2)^{0.5} e^{a\delta} \cos(b\delta + \tan^{-1}(q/p)) \\ q' &= (p^2+q^2)^{0.5} e^{a\delta} \sin(b\delta + \tan^{-1}(q/p)) \end{aligned}$$

We wish to find linear constraints that are implied by the above equation based on analyzing the phase.

Given p', q', let $\omega(p', q')$ denote the angle $b\delta + \tan^{-1}(q/p)$. For example, $\omega(p, q)$ is just $b(t-t) + \tan^{-1}(q/p) = \tan^{-1}(q/p)$. We have the following relationship based on analyzing the phase.

$$b\delta = \omega(p', q') - \omega(p, q)$$

Now, we need piecewise linear approximations of the ω function. The main point to note is that the phase determines the sign of p', q' and the value of $p' \geq q'$. Hence, we can get an estimate of the phase of p, q if we analyze the signs of $p, q, p-q$. Depending on the sign of $p, q, p-q$, the time axis is partitioned into infinitely many intervals, as shown in Figure 5.2.

Let us define the function $\omega_{\mathsf{a}}(p, q)$ that takes the values of p, q and returns a number based on the phase as illustrated in Figure 5.2.

$$\omega_{\mathsf{a}}(p,q) \;=\; \begin{cases} 0 & \text{if } p > q \geq 0 \\ 1 & \text{if } q \geq p > 0 \\ 2 & \text{if } q > -p \geq 0 \\ 3 & \text{if } -p \geq q > 0 \\ 4 & \text{if } -p > -q \geq 0 \\ 5 & \text{if } -q \geq -p > 0 \\ 6 & \text{if } -q > p \geq 0 \\ 7 & \text{if } p \geq -q > 0 \end{cases}$$

Now, given $\omega_{\mathsf{a}}(p, q)$ and $\omega_{\mathsf{a}}(p', q')$, we can compute bounds on δ. Specifically, if $\omega_{\mathsf{a}}(p', q') \geq \omega_{\mathsf{a}}(p, q)$, then for some natural number $n \geq 0$, it will be the case that

$$\begin{aligned} &\phi^{\mathtt{phase}}_{\langle \lambda, \vec{c} \rangle} := \exists n \geq 0 : \\ &b\delta \;\geq\; 2\pi n + ((\omega_{\mathsf{a}}(p', q') - \omega_{\mathsf{a}}(p, q)) - 1)\frac{2\pi}{8} \quad \wedge \\ &b\delta \;\leq\; 2\pi n + ((\omega_{\mathsf{a}}(p', q') - \omega_{\mathsf{a}}(p, q)) + 1)\frac{2\pi}{8} \end{aligned} \tag{5.12}$$

The value of n indicates the number of complete cycles that lie between the initial and final state.

Similarly, if $\omega_{\mathsf{a}}(p', q') \leq \omega_{\mathsf{a}}(p, q)$, then for some natural number $n \geq 0$, it will be the case that

$$\begin{aligned}
&\phi^{\texttt{phase}}_{\langle\lambda,\vec{c}\rangle} := \exists n \geq 0 : \\
b\delta &\geq 2\pi n + ((\omega_{\texttt{a}}(p',q') - \omega_{\texttt{a}}(p,q)) + 7)\frac{2\pi}{8} \quad \wedge \\
b\delta &\leq 2\pi n + ((\omega_{\texttt{a}}(p',q') - \omega_{\texttt{a}}(p,q)) + 9)\frac{2\pi}{8}
\end{aligned} \tag{5.13}$$

In both cases, the constraint $\phi^{\texttt{phase}}_{\langle\lambda,\vec{c}\rangle}$ is an infinite disjunction: there is one disjunct for each value of n. There are two different ways to handle the infinite disjunction. First, in the time-aware relational abstraction, we can introduce a new *input* variable n. When we perform infinite bounded model checking on the abstract system, the input variable n is automatically existentially quantified and all possible values for n are considered. The alternative is to replace the infinite disjunction by a finite disjunction (picking specific values for n, say $n = 0, 1, 2$) and then over-approximating the rest ($n = 3, 4, \ldots$) of the disjuncts conservatively. Our implementation uses the latter approach. We have a parameter that fixes the range of values we use for n. We will call the parameter n subsequently.

5.3.5 Correctness

We can now formally state the correctness of the procedure outlined above for creating a time-aware relational abstraction. First, we note the following immediate fact.

Lemma 1 *If $\phi_1(\vec{x}, \vec{x}')$ and $\phi_2(\vec{x}, \vec{x}')$ are two relational abstractions of the same system, then $\phi_1(\vec{x}, \vec{x}') \wedge \phi_2(\vec{x}, \vec{x}')$ is also a relational abstraction of that system.*

Thus, for a given linear system $\dot{\vec{x}} = A\vec{x} + \vec{b}$, let Λ denote all pairs $\langle\lambda, \vec{c}\rangle$ s.t. $\frac{d\vec{c}^T\vec{x}}{dt} = \lambda\vec{c}^T\vec{x}$, and let R denote all pairs $\langle r, \vec{c}\rangle$ s.t. $\frac{d\vec{c}^T\vec{x}}{dt} = r$ for some real number r. Then, we can construct the following time-aware relational abstraction for $\dot{\vec{x}} = A\vec{x} + \vec{b}$:

$$\textsc{Timed}^t_q \;:=\; \bigwedge_{\langle\lambda,\vec{c}\rangle\in\Lambda} \phi^t_{\langle\lambda,\vec{c}\rangle} \;\wedge\; \bigwedge_{\langle r,\vec{c}\rangle\in R} \phi^c_{\langle r,\vec{c}\rangle} \tag{5.14}$$

where depending on whether λ is real or complex, $\phi^t_{\langle\lambda,\vec{c}\rangle}$ is defined in Equation 5.5 or Equation 5.10, and $\phi^c_{\langle r,\vec{c}\rangle}$ is defined in Equation 5.3.

The following theorem states the correctness of this construction.

Theorem 9 *Let $\dot{\vec{x}} = A\vec{x}+\vec{b}$ be the continuous dynamics of the location $q \in Q$ of H. Then, $\textsc{Timed}^t_q$ defined in Equation 5.14 is a relational abstraction for the continuous dynamics $Flow(q)$.*

Proof. We have to prove that $\forall v_i, v_{i+1} \in \mathbb{R}^n, \delta \in \mathbb{R}$ s.t. $v_{i+1} = f_q(v_i, \delta)$, $s, s' \models \textsc{Timed}^t_q$, where $s = \rho(\langle q, v_i\rangle)$, $s' = \rho(\langle q, f(v_i, \delta)\rangle)$ and $f_q : \mathbb{R}^{n+1} \to \mathbb{R}^n$ is the solution to the flow condition $Flow(q)$.

We will prove that s, s' is a model for each conjunct $\phi^t_{\langle\lambda,\vec{c}\rangle}$ and $\phi^c_{\langle r,\vec{c}\rangle}$. Then, the proof will follow from Lemma 1.

The proof for the constant rate case, i.e. $s, s' \models \phi^c_{\langle r,\vec{c}\rangle} := p' - p = r\delta$, follows directly from the fact that $\dot{p} = r$. Note that $\dot{t} = 1$, hence $\delta = \delta$.

We prove that $s, s' \models \phi^t_{\langle\lambda,\vec{c}\rangle}$ considering the cases for real and complex eigenvalues.

Suppose $\lambda \in \mathbb{R}$ and $\vec{c} \in \mathbb{R}^n$. If $\lambda = 0$, clearly $s, s' \models p' = p$. If $\lambda > 0$, then $\phi^t_{\langle\lambda,\vec{c}\rangle} = \psi(p, p')$ for the predicate $p = \vec{c}^T\vec{x}$. We have that $s, s' \models \psi(p, p')$:

1. If $s \models p = 0$, then by the explicit time solution of p (Equation A.1), we also have $p' = 0$ and thus $s, s' \models 0 = p = p'$.

2. If $s \models 0 < p$, then by Equation A.1 we have $s, s' \models 0 < p \leq p'$ and $s, s' \models \ln(p') - \ln(p) = \lambda\delta$. Thus, $s, s' \models \ln_{\texttt{lb}}(p') - \ln_{\texttt{ub}}(p) \leq \lambda\delta \leq \ln_{\texttt{ub}}(p') - \ln_{\texttt{lb}}(p)$.

3. The proof for the other case (i.e. when $s, s' \models 0 < p \leq p'$) is similar to the previous one.

The proof for $\lambda < 0$ is specular to the one for $\lambda > 0$.

Suppose $\vec{c} = \vec{d} + \iota\vec{e}$ is a left eigenvector of A corresponding to the complex eigenvalue $\lambda = a + \iota b$. In this case, we have that $\phi^{t}_{\langle\lambda,\vec{c}\rangle} := \phi^{\texttt{ampl}}_{\langle\lambda,\vec{c}\rangle} \wedge \phi^{\texttt{phase}}_{\langle\lambda,\vec{c}\rangle}$

We can prove that $\phi^{\texttt{ampl}}_{\langle\lambda,\vec{c}\rangle}$ is a relational abstraction following the same reasoning done for the real-eigenvalue case (i.e. the explicit solution now is the one in Equation 5.11).

To prove that $s, s' \models \phi^{\texttt{phase}}_{\langle\lambda,\vec{c}\rangle}$, we have to consider all the possible combinations of intervals $\omega_{\texttt{a}}(p, q)$ and $\omega_{\texttt{a}}(p', q')$ in the case $\omega_{\texttt{a}}(p, q) \leq \omega_{\texttt{a}}(p', q')$ or $\omega_{\texttt{a}}(p', q') \leq \omega_{\texttt{a}}(p, q)$. We show the proof for one case, while the others can be proved similarly, hence proving that $s, s' \models \phi^{\texttt{phase}}_{\langle\lambda,\vec{c}\rangle}$.

Suppose that $s \models 0 \leq q \leq p$ and $s' \models 0 \leq q' \leq p'$. Thus, $\omega_{\texttt{a}}(p, q) = \omega_{\texttt{a}}(p', q') = 0$. From $s' \models 0 \leq q' \leq p'$ and the solution of p' and q' (Equation 5.12) we have

$$0 \leq \cos(b\delta + \tan^{-1}(q/p)) \leq \sin(b\delta + \tan^{-1}(q/p))$$

We derive a lower and upper bound for $b\delta + \tan^{-1}(q/p)$:

$$2\pi k_1 \leq b\delta + \tan^{-1}(q/p) \leq 2\pi k_1 + \frac{\pi}{4}$$

for some integer k_1. From $0 \leq q \leq p$, we have that:

$$2\pi k_2 \leq \tan^{-1}(q/p) \leq 2\pi k_2 + \frac{\pi}{4}$$

for some integer k_2. Thus, we have:

$$2\pi(k_1 - k_2) - \frac{\pi}{4} \leq b\delta \leq 2\pi(k_1 - k_2) + \frac{\pi}{4}$$

Letting $n = k_1 = k_2$, we obtain the abstraction $\phi^{\texttt{phase}}_{\langle\lambda,\vec{c}\rangle}$ for $\omega_{\texttt{a}}(p, q) = \omega_{\texttt{a}}(p', q') = 0$. $\diamond$

5.4 Related Work

Verification of hybrid systems has been performed applying different techniques (see [Alu11] for a recent survey).

Among these techniques there are symbolic reachability and deductive verification. Symbolic reachability [HHWT97, FGD$^+$11b, TK04, BBC$^+$12, BBC$^+$14, RS07] consists of computing the reachable set of states, which will be used to prove the system safety. In deductive verification [Pla08, PJ04, ST11b], the user interacts with a theorem prover to produce a proof of correctness. Both approaches may handle properties that involve predicates over time. However, we stress that the main goal of the time-aware abstraction is to provide a more precise relational abstraction, widening its applicability in terms of hybrid systems and properties that can be verified. Also, since the timed-aware abstraction is a relational abstraction, it separates the reasoning task on the continuous dynamics from the verification task on the infinite-state transition system.

Another viable technique to the hybrid systems verification consists of computing an abstraction of the system, which may be subsequently verified [ADI06, CFH$^+$03, Tiw08]. Relational abstraction falls in this class of techniques. There exists several ways to compute relational abstractions. However, the current techniques used to compute a relational abstraction [ST11a, Tiw12, ZST12] are not suitable to verify real-time properties or to analyze a network of hybrid systems.

The template-based relational abstraction [ST11a] does not capture the relation among time and the continuous variables of the system. Moreover, the problem of finding the coefficients of the templates requires the use of non-linear real arithmetic, which may be solved using expensive real quantifier elimination techniques [DS96, Bro03] or SMT solvers which handle non-linear real arithmetic [JdM12].

The timed relational abstraction [ZST12] is suitable to analyze control systems that sample the physical plant at fixed time intervals. In that case, the relation is precise over time since it relates the current values with the future values of the variables after a continuous transition of fixed duration (the sampling interval). However, since the continuous evolution has a fixed duration, the relation does not capture the possible evolution of the system for different intervals of time.

As explained in Section 5.2, the eigenstructure-based relational abstraction [ST11a, Tiw12] is not precise enough to capture the relation between the continuous variables and the time elapsed in a continuous transitions and, in general, between the variables which evolve with different "rates". Hence, it is not suitable to analyze properties which predicate about time or about other variables with a piecewise constant derivative (like drifted clocks or resources which evolve with a non-deterministic but bounded derivative). The time-aware abstraction increases the precision of the eigenstructure-based abstraction.

Other works [DM12] use relational abstraction to capture a sequence of continuous and discrete transitions, with the goal of verifying stability properties. The generated abstraction is expressed in non-linear real arithmetic and it does not explicitly consider the relation with time.

Other works focuses on the analysis of hybrid systems using Satisfiability Modulo Theory (SMT) solvers [ABCS05, ÁBKS05]. However, these approaches are currently limited, since they may only handle a subset of Linear Hybrid Systems [CMT12, CMT13b]. Relational abstraction handles all the class of linear hybrid systems, even if in an approximate way. The time-aware abstraction also widens the applicability of the verification techniques developed for networks of linear hybrid automata, like scenario verification [CMT13c], to Linear Hybrid Systems.

Since the produced abstraction is an infinite-state transition system, it can be verified by SMT-based verification techniques such as k-induction or ic3 [SSS00, HB12, CG12]. The time-aware abstraction is orthogonal to these approaches, since it only abstracts a dynamical system. Our approach will benefit from any improvement in the performance of the verification algorithms for infinite state transition systems and of SMT solvers.

Part III

Verification Techniques

Chapter 6

Reachability

In this chapter we present two techniques based on Satisfiability Modulo Theories to solve the *Reachability Problem*.

The first technique, *Shallow Synchronization* [BCL$^+$10a] is a specialized Bounded Model Checking encoding for a network of hybrid automata. The encoding is such that the paths of each automaton in the network are explored independently, while their consistency is ensured by additional constraints, which force synchronization actions to happen at the same time (as in local-time semantic), but allow synchronizations to happen at different steps in the encoding. Note that this technique may find a counterexample to an invariant property, and not prove that the property holds.

The second technique is a verification algorithm based on predicate abstraction [GS97] and K-induction [SSS00]. The technique, called *Implicit Abstraction* [Ton09], embeds predicate abstraction in the symbolic encoding of paths of BMC (and thus also of K-induction). The result is that predicate abstraction is not computed beforehand, but on demand while solving the K-induction problem. Related to abstraction, we show that it can be refined automatically adding predicates extracted from interpolants [HJMM04]. Differently from Bounded Model Checking, the algorithm may prove that a property holds. Also, note that abstraction is crucial for infinite-state transition systems, where the simple path condition of K-induction may be ineffective. While the technique works in general for symbolic transition systems, it could be applied to hybrid systems using the encodings presented in Part II.

First, we formally introduce the reachability problem in Section 6.1. Then, in Section 6.2 we present *Shallow synchronization* and finally in Section 6.3 the combination of k-induction and predicate abstraction.

6.1 Problem Definition

In the following, we consider the reachability problem for a network of hybrid automata.

Definition 19 (Reachability Problem) *Given a network of hybrid automata $\mathcal{N} = H_1 || \ldots || H_n$ and a state s of $\mathcal{N}$, the reachability problem consists of deciding if there exists a finite path π of $\mathcal{N}$ that reaches s (i.e. $\pi = s_0 \xrightarrow{\delta_1} s_1 \xrightarrow{\delta_2} \ldots \xrightarrow{\delta_k} s$).*

The problem can be extended to a set of states S: we say that S is reachable if there exists a state $s \in S$ that is reachable.

Given a network of hybrid automata $\mathcal{N}$ and a formula P defined over variables $\bigcup_{1 \leq i \leq n} V_i \cup X_i$, we say that $\mathcal{N}$ satisfy P, $\mathcal{N} \models P$, if $\neg P$ is not reachable.

6.2 Bounded Model Checking with Shallow Synchronization

Note. The material presented in this chapter has already been presented in [BCL$^+$10a]. The traditional asynchronous semantics is based on interleaving, and requires the construction of a monolithic hybrid automaton based on the composition of the automata in the network. Intuitively, this means that a path in the automaton is the result of the composition of interleaving paths. However, the monolithic automaton resulting from the composition can be seen as the result of a “strict synchronization”, forcing the analysis to deal with a large number of paths, where the structure and the locality of the network are not taken into account.

An alternative semantics [BL11] for networks of automata exploits the fact that automata can be “shallowly synchronized”. The intuition is that each automata can proceed based on their individual “local time scale”, unless they perform a synchronizing transition, in that case they must realign their absolute time. This results in a more concise semantics, where traces of the network are obtained by composing traces of local automata, each with local time elapse, forcing their consistency with respect to the shared communication.

We provide a Bounded Model Checking encoding that exploits “shallow synchronization”, exploring various search strategies. The main advantage of the approach is that the transition relation of each automata is unrolled only for the steps necessary to reach locally the target (regardless the length of the interleaving with the other automata). Typically, local paths are much shorter because they do not need to stutter allowing other processes to perform local or non-shared events. The disadvantage is that we may use additional variables and constraints that degrades the performance of the approach.

6.2.1 Shallow Synchronization Semantics

While in strict synchronization the behavior of a network is basically obtained by interleaving, in shallow synchronization a path of the network is the result of the “composition” of paths local to each automaton in the network. The intuition is demonstrated in Figure 6.1. In the upper part,

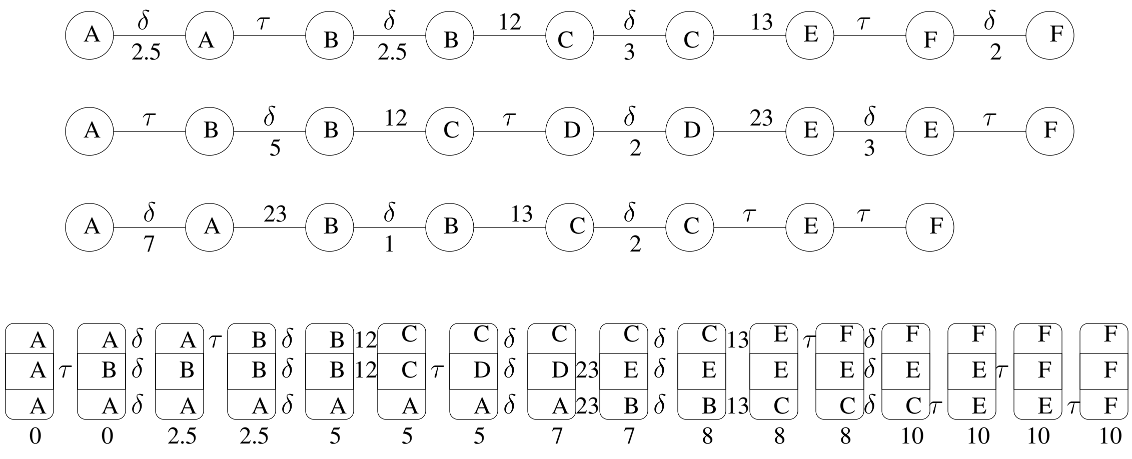

A δ 2.5 A τ B δ 2.5 B 12 C δ 3 C 13 E τ F δ 2 F
A τ B δ 5 B 12 C τ D δ 2 D 23 E δ 3 E τ F
A δ 7 A 23 B δ 1 B 13 C δ 2 C τ E τ F
0 0 2.5 2.5 5 5 5 7 7 8 8 8 10 10 10 10

Figure 6.1: Three local traces (above), and the corresponding interleaving (below).

we see three traces of three automata in a network. Each automaton H_i has a local label τ; the ij labels are shared between processes H_i and H_j; δ denotes the local time elapse. We notice that the synchronization over the ij labels happens exactly at the same time, e.g., 12 takes place at absolute time 5, although the number of transitions required by H_1 and H_2 is different. In the lower part of the figure, we report the corresponding trace based on interleaving (where each box contains the state of each of the three processes). Stuttering (e.g. of process 1 and 3 in the first step) is modeled by the fact that a process does not have any label on its side.

In the following, we consider a network of hybrid automata $\mathcal{N} = H_1 || \ldots || H_n$ and an invariant property P. We associate to each automaton H_i of $\mathcal{N}$ a transition system S_i, as we did in Section 4.2 when defining the local-time encoding. S_i is defined as in the local-time case, adding an additional variable t_i to keep track of the total amount of time elapsed in the system.

We define a mapping of a set of shallowly synchronized paths of the transition systems $S_1, \ldots, S_n$ into a path in the composition $S_{\textsc{LocTime}}(\mathcal{N})$. Intuitively, the mapping induces an equivalence relation among the paths of $S_{\mathrm{LocTime}}(\mathcal{N})$ which are obtained by composing the same set of local paths with different interleavings. The shallow synchronization is defined according to the trace of a path i.e., the list of events occurring in the path. An S-trace, with $S \subseteq A$, is a trace restricted to the labels in the set S.

Definition 20 (S-trace) *Given a set of events $S \subseteq A$ and a path $\pi = s_0; a_1; s_1; \ldots; a_h; s_h$, the S-*trace *$\tau_S(\pi)$ is the sequence of events $\langle t_1, a_1\rangle; \ldots; \langle t_k, a_k\rangle$ where t_j is the time at which the event a_j occurs in π and $a_j \in S$, for $1 \leq j \leq k$.*

Definition 21 (Consistent traces) *Let π_1 and π_2 be two paths over the sets of events A_1 and A_2 respectively, and $S = A_1 \cap A_2$. The pair $\langle \pi_1, \pi_2\rangle$ is* consistent *iff the S-trace of π_1 is equal to the S-trace of π_2 ($\tau_S(\pi_1) = \tau_S(\pi_2)$) and the final time of π_1 is equal to the final time of π_2.*

The last constraint on the final time is necessary because otherwise the two paths may terminate with a series of local steps with different timings.

Definition 22 (Shallowly synchronized path) *A* shallowly synchronized path *of a network $\mathcal{N}$ is a tuple $\pi_{\mathrm{SHALLOW}} = \langle \pi_1, \ldots, \pi_n\rangle$ such that, for all $1 \leq i \leq n$, π_i is a path of S_i and, for all i, j, $1 \leq i < j \leq n$, π_i and π_j are consistent.*

If π is a shallowly synchronized path, we denote with π_i the i-th component of π.

Remark 4 *In general, two different events can occur at the same time in the same path, because discrete transitions are not forced to be interleaved with timed transitions. Moreover, simultaneous events may be interleaved with different orders.*

However, in many cases, we can assume that whenever two events occur simultaneously, they have a fixed order. Then, the pair $\langle \pi_1, \pi_2\rangle$ is consistent *simply iff for all $a \in A_1 \cap A_2$ and $t \in \mathbb{R}$, $\langle a, t\rangle$ occurs in π_1 iff $\langle a, t\rangle$ occurs in π_2. I.e., having the events at the same time guarantees that the traces are the same. The definitions and theorems in [BL11] have this assumption, while in this section we consider the most general case.*

The projection of a synchronized path on a specific transition system S_i is the local path of that transition system S_i. Intuitively, the set of projections of a synchronized path form a shallowly synchronized path. The projection induces an equivalence relation over strictly synchronized traces, namely the equivalence of paths that are the same modulo a reordering of the interleaved labels.

Definition 23 (Projection) *Given the transition system S_i and a path $\pi_{\textsc{LocTime}}$ in $S_{\textsc{LocTime}}(\mathcal{N})$, the projection of $\pi_{\textsc{LocTime}}$ over S_i is the path $prj(\pi_{\textsc{LocTime}}, i)$ obtained projecting the states over the S_i-th component and removing all the transitions over events which are not in $A_i \cup \{\textsc{t}_i\}$ (in the alphabet of H_i and the timed event). Note that the stutter event ($\textsc{s}$) is projected. Formally, given the component S_i with alphabet A_i and $\pi_{\textsc{LocTime}} := s_0; a_1; s_1; \ldots; a_h; s_h$, the projection of $\pi_{\textsc{LocTime}}$ over S_i is the path $prj(\pi_{\textsc{LocTime}}, i) := s'_0; a'_1; s'_1; \ldots; a'_l; s'_l$ such that:*

- *$f_{A_i} : \mathbb{N}_{\geq 0} \times \mathbb{N}_{\geq 0}$ is a function such that $f_{A_i}(z)$ maps the index in $\pi_{\textsc{LocTime}}$ of the z-th occurrence of an event which belongs to the set $A_i \cup \{\textsc{t}_i\}$.*
- *l is the number of events in the path $\pi_{\textsc{LocTime}}$ which also belong to $A_i \cup \{\textsc{t}_i\}$.*
- *$s'_0 := s_0$.*
- *for all $1 \leq j \leq l$, $s'_j := s_{f_{A_i \cup \{\textsc{t}_i\}}(j)}{}_{|V_i}$.*
- *for all $1 \leq j \leq l$, $a'_j := a_{f_{A_i \cup \{\textsc{t}_i\}}(j)}{}_{|A_i}$.*

The following theorem states the relationship between the local-time semantic and shallow synchronization[1].

Theorem 10 *If* $\pi_{\text{LocTime}} := s_0; a_1; s_1; \ldots; a_h; s_h$ *is a path in the local-time semantics then* $\pi_{\text{SHALLOW}} = \langle prj(\pi_{\text{LocTime}}, 1), \ldots, prj(\pi_{\text{LocTime}}, n)\rangle$ *is a* shallowly synchronized *path.*
Vice versa, given a shallowly synchronized *path* π_{SHALLOW} *there exists a path* π_{LocTime} *in* $S_{\text{LocTime}}(\mathcal{N})$ *such that* $\pi_{\text{SHALLOW}} = \langle prj(\pi_{\text{LocTime}}, 1), \ldots, prj(\pi_{\text{LocTime}}, n)\rangle$.

Proof. ($\Rightarrow$) π_{SHALLOW} is a *shallowly synchronized path* since:

- for all $1 \leq i \leq n$, $prj(\pi_{\text{LocTime}}, i) \models S_i$: By construction, the projection of $prj(\pi_{\text{LocTime}}, i)$ removes from the network path π_{LocTime} all the transitions which are not over $A_i \cup \{\text{T}_i\}$. $prj(\pi_{\text{LocTime}}, i) \models S_i$, since in the transitions removed from π_{LocTime} S_i stutters, thus it does not change the value of its local states.

- for all $1 \leq i < j \leq n$, $prj(\pi_{\text{LocTime}}, i)$ and $prj(\pi_{\text{LocTime}}, j)$ are consistent: by construction, the projection does not change the order of states and events of π_{LocTime}, and restricts each projection to a given alphabet. $prj(\pi_{\text{LocTime}}, i)$ is restricted to all the events in $A_i \cup \{\text{T}_i\}$ while $prj(\pi_{\text{LocTime}}, j)$ is restricted to all the events in $A_j \cup \{\text{T}_j\}$. Consider the projection of $prj(\pi_{\text{LocTime}}, i)$ and $prj(\pi_{\text{LocTime}}, j)$ over the common set of events of $A_i \cap A_j$. The two sequences contain the same events and have the same order by hypothesis and, therefore, they are equal. Moreover, since π_{LocTime} is in $S_{\text{LocTime}}(\mathcal{N})$, the assignment to the variables t_i, t_j in the last state of π_{LocTime} is the same. Thus, also

[1]In the extended version [BCL+10b] of [BCL+10a] the theorem relates paths of the global-time semantics with shallowly synchronized paths, instead of considering the local-time semantics. Applying Theorem 8 we can get the same result.

the assignments to t_i and t_j in the last state of $prj(\pi_{\text{LocTime}}, i)$ and $prj(\pi_{\text{LocTime}}, j)$ respectively must be the same.

($\Leftarrow$) If π_{SHALLOW} is a *shallowly synchronized* path, then there exists a path π_{LocTime} in $S_{\text{LocTime}}(\mathcal{N})$ such that $\pi_{\text{SHALLOW}} = \langle prj(\pi_{\text{LocTime}}, 1), \ldots, prj(\pi_{\text{LocTime}}, n)\rangle$. $\pi_i := s^i_0; a^i_1; s^i_1; \ldots; a^i_{l^i}; s^i_{l^i}$. We recursively define the function γ which maps π_{SHALLOW} to a path $\pi_{\text{LocTime}} \in S_{\text{LocTime}}(\mathcal{N})$ ($\pi_{\text{LocTime}} := \gamma(\pi_{\text{SHALLOW}})$):

- *Base case:* if $\pi_i = s^i_0$ for all $1 \leq i \leq n$, then $\gamma(\pi_{\text{SHALLOW}}) := s^1_0, \ldots, s^n_0$ (i.e. all the local paths have a single state).

- *Local transition:* if there exists an i in $1 \leq i \leq n$ such that $\pi_i := s^i_0; a^i_0; \ldots; s^i_{l^i-1}; a^i_{l^i}; s^i_{l^i}$ and a^i_0 is a local event of S_i, then $\gamma(\pi_{\text{SHALLOW}}) := s^1_0, \ldots, s^i_0, \ldots, s^n_0;\ a^1, \ldots, a^n;\ \gamma(\langle \pi_1, \ldots, \pi'_i, \ldots, \pi_n\rangle)$, where

$$a^j = \begin{cases} a^j_0 & \text{if } i = i \\ a^j \text{ s.t. } a^j \models \varepsilon_j = \text{S} & \text{otherwise} \end{cases}$$

  and $\pi'_i := s^i_1; a^i_1; \ldots; s^i_{l^i-1}; a^i_{l^i}; s^i_{l^i}$ (i.e. when a process can move on a local event, all the other processes stutter).

- *Shared transitions:* otherwise, since π_{SHALLOW} is a *shallowly synchronized path*, there exists a set J of indexes and and event $a \in \bigcap_{i\in J} A_i$ such that for all $i \in J$, $a_i \models \varepsilon_i = a$, and $\bigwedge_{i,z\in J} s_i(t_i) = s_z(t_z)$ (i.e. all the processes with index in J synchronize on the event a). In this case $\gamma(\pi_{\text{SHALLOW}}) := s^1_0, \ldots, s^i_0, \ldots, s^n_0;\ a^1, \ldots, a^n;\ \gamma(\langle \pi'_1, \ldots, \pi'_n\rangle)$, where:

$$a^i = \begin{cases} a^i_0 & \text{if } i \in J \\ a^i \text{ s.t. } a^i \models \varepsilon_i = \text{S} & \text{otherwise} \end{cases}$$

  and

$$\pi'_i = \begin{cases} s^i_1; a^i_1; \ldots; s^i_{l^i-1}; a^i_{l^i}; s^i_{l^i} & \text{if } i \in J \\ \pi_i & \text{otherwise} \end{cases}$$

We can prove by induction that $\pi_{\text{LocTime}} \in S_{\text{LocTime}}(\mathcal{N})$. Moreover, since the last time of all the components in $\gamma(\pi_{\text{SHALLOW}})$ are equal, then $\langle prj(\pi_{\text{LocTime}}, 1), \ldots, prj(\pi_{\text{LocTime}}, m)\rangle$ is a *shallowly synchronized path.* $\diamond$

6.2.2 Symbolic Encoding

In this section we first recall the Bounded Model Checking encoding for the individual symbolic transition systems $S_1, \ldots, S_n$, and then the monolithic encoding for the network $S_{\text{LocTime}}(\mathcal{N})$. Then, we show how we can encode symbolically the problem exploiting shallow synchronization.

Given a transition system $S_i = \langle V_i, \mathcal{W}_i, Init_i, Inv_i, Trans_i\rangle$, its bounded model checking encoding at depth k is defined as in Section 2.3.1:

$$BMC_{S_i}(k) \;:=\; Init_i^0 \wedge \bigwedge_{j=0}^{k} Inv_i^j \wedge \bigwedge_{j=0}^{k-1} Trans_i^j \wedge \neg P^k$$

$BMC_{S_i}(k)$ is satisfiable iff there exists a path reaching the target condition in k steps.

Similarly, we can encode the reachability problem for a network $\mathcal{N}$ using the monolithic transition system $S_{\text{LocTime}}(\mathcal{N}) = \langle V, \mathcal{W}, Init, Inv, Trans\rangle$:

$$BMC_{S_{\text{LocTime}}(\mathcal{N})}(k) \;:=\; Init^0 \wedge \bigwedge_{i=0}^{k} Inv^i \wedge \bigwedge_{i=0}^{k-1} Trans^i \wedge \neg P^k$$

Also the encoding in the case of global-time encoding $S_{\text{GLTime}}(\mathcal{N})$ is similar.

In both cases, the encoding is "compositional" in the sense that each automaton is individually encoded. However, the necessity of stuttering on non-shared events and of performing shared events in the same steps may cause complex paths (as shown in Fig. 6.1).

In the experimental evaluation, we also consider the variant of the above encoding where we allow discrete transitions in different automata to occur at the same step of the encoding (e.g. the step semantic variant of the synchronization constraints).

Now, we show the BMC encoding based on shallow synchronization. We allow to unroll each transition system for a different bound k, we do not force processes to stutter and we let shared events to occur at different (local) steps. This means that each of the local encodings is able to construct a local trace.

The reachability problem with bounds $\overline{k} = \langle k_1, k_2, \ldots, k_n \rangle$ can be encoded as

$$\textsc{BmcSS}(\overline{k}) \;:=\; \bigwedge_{1 \leq i \leq n} BMC_{S_i}(k_i) \wedge \textsc{ShallowSync}$$

where ShallowSync encodes the constraints enforcing that all the paths must be consistent according to Definition 21. In the following, we present different ways to encode ShallowSync: an encoding based on the enumeration of events, and encoding based on local reasoning and one variant that exploits the Theory of Equalities and Uninterpreted Functions (we assume to be in the case described in Remark 4, but all the encodings that we are showing can be lifted to the general case.)

Encoding based on enumeration

The first way to encode ShallowSync is by enumerating all possible combinations of steps on which the synchronization occurs. For example, processes P1 and P2 may synchronize over event a, but a may occur in step 2 for P1, and in step 4 for P2. ShallowSync guarantees that, for all pairs of processes, (i) if a shared event occurs in the first process, then the event must occur also in the second process at the same time (possibly in different steps), and (ii) the final time of the two processes is the same.

The encoding of ShallowSync is:

$$\bigwedge_{1\leq j<h\leq n}\ \bigwedge_{a\in A_j\cap A_h} \bigwedge\nolimits_{1\leq i_j\leq k_j}(\varepsilon_j^{i_j}=a \leftrightarrow \bigvee\nolimits_{1\leq i_h\leq k_h}\varepsilon_h^{i_h}=a \wedge t_j^{i_j}=t_h^{i_h})\ \wedge$$

$$\bigwedge\nolimits_{1\leq i_h\leq k_h}(\varepsilon_h^{i_h}=a \leftrightarrow \bigvee\nolimits_{1\leq i_j\leq k_j}\varepsilon_j^{i_j}=a \wedge t_j^{i_j}=t_h^{i_h})\ \wedge$$

$$\bigwedge_{1<j\leq n} t_j^{k_j}=t_1^{k_1}$$

Local reasoning

We propose a variant of the previous encoding which can be split into constraints local to each automaton, and one for each step. The BMC encoding uses several additional variables. We add the variables to the BMC encoding and not to the symbolic transition systems. The additional variables are:

- for each transition system S_j, for each shared event a (i.e. $a \in \bigcup_{1\leq z\leq n, z\neq j}(A_j\cap A_z)$), for each step of the encoding $i \leq k_j$, we declare an additional variable $count_{a,j}^i$ to represent how many times a has occurred in S_j before step i;
- for each shared label a (i.e. $a \in \bigcup_{1\leq z\leq n, z\neq j}(A_j\cap A_z)$), a group of variables $occ\_time_{i,a}$ to represent the time at which the i-th occurrence of a is fired;
- for each shared label a, (i.e. $a \in \bigcup_{1\leq z<j\leq n}(A_j\cap A_z)$) a variable a_{last} to record how many times a has been fired in the whole path;
- a variable c_{last} to record the time at which the system reaches the target.

We may encode all the newly introduced counter variables using different domains, like bounded integers (they could be bounded by the maximum depth we want to reach) or reals. Also, note that the variables without superscript are untimed, in the sense that they do not depend on any temporal step.

The shallow synchronization formula $\textsc{ShallowSync}$ can be encoded as:

$$\bigwedge_{1 \leq j \leq n} \left(\bigwedge_{0 \leq i < k_j} (\textsc{ShallowStep}_j^i) \wedge \textsc{CounterInit}_j \wedge \right. \\ \left. \bigwedge_{0 \leq i < k_j} \textsc{CounterStep}_j^i \wedge \textsc{FinalShallow}_j \right)$$

where $\textsc{ShallowStep}_j^i$ states that if in the i-th step, an event a occurs in the j-th process for the g-th time, then the local time of the process must be $occ\_time_{g,a}$:

$$\textsc{ShallowStep}_j^i := \bigwedge_{a \in A_j} (\varepsilon_j^i = a \to \bigwedge_{1 \leq g \leq i} (count_{a,j}^i = g \to t_j^i = occ\_time_{g,a}))$$

$\textsc{CounterInit}$ and $\textsc{CounterStep}$ encode how the counters evolve:

$$\begin{aligned} \textsc{CounterStep}_j^i &:= (\bigwedge_{a \in A_j} (\varepsilon_j^i = a \to count_{a,j}^{i+1} = count_{a,j}^i + 1)) \wedge \\ &\quad ((\varepsilon_j^i = \textsc{s} \vee \varepsilon_j^i = \textsc{t}) \to (count_{a,j}^{i+1} = count_{a,j}^i)) \\ \textsc{CounterInit}_j &:= (count_{a,j}^0 = 0) \end{aligned}$$

while $\textsc{FinalShallow}$ states that the final values of the counters and the local time must be the same:

$$\textsc{FinalShallow}_j := (\bigwedge_{a \in A_j} count_{a,j}^{k_j} = a_{last}) \wedge (t_j^{k_j+1} = c_{last}))$$

Exploiting richer theories It is possible to represent the above encoding with richer theories introducing uninterpreted functions symbols. In particular we represent the time of the i-th occurrence of a label a as a function *occ_time*$_a$ from integers to reals. This way we can rewrite ShallowStep into

$$\textsc{ShallowStep}^i_j := \bigwedge_{a \in A_j} (\varepsilon^i_j = a) \rightarrow (t^i_j = \mathit{occ\_time}_a(\mathit{count}^i_{a,j}))$$

6.2.3 Related Work

The shallow synchronization encoding can be used to falsify an invariant property (i.e. proving that a set of states is reachable), There exists several approaches that focus on the falsification problem for hybrid systems [NMA$^+$02, ACKS02, Sor02, dMRS02, BZL10, ERNF11, IUH11, GKC13, PKV13, NSF$^+$10, MN10, BL11, CMT13b, BLW$^+$10].

Some approaches [NMA$^+$02, ACKS02, Sor02, dMRS02, ERNF11, IUH11, GKC13, CMT13b] are based on Bounded Model Checking and SMT solvers. The characterizing feature of our work is the attempt to leverage the structure induced by the synchronization of a network of hybrid automata.

The first approach that adopts a shallowly synchronized semantics is presented in [BL11] for path-oriented bounded reachability analysis of a network of LHAs. In the approach, one path is selected for each component and all selected paths compose a path set for reachability analysis. Each path is independently encoded to a set of constraints while synchronization controls are encoded according to the position of shared labels. By merging all the constraints, the path-oriented reachability problem can be transformed to the feasibility problem of the resulting linear constraint set, which can be solved by linear programming efficiently. This approach has been extended in BACH [BLW$^+$10] into a general bounded reachability analysis technique. Differently from our approach, this technique traverses the structure of a network of automata using depth-first search and checks the abstract path set one by one.

In the approaches mentioned above, the search is carried out in two stages: in the first, a discrete abstraction of the problem is constructed, while in the second the candidate paths found in the abstract state are checked for consistency in the concrete space. In our approach, the SMT solver carries out the refinement automatically during the search, on demand. With respect to explicit-state search, the symbolic representation is less sensitive to the state-space explosion problem. With respect to abstraction-based techniques, the BMC technique is more tailored to find error paths.

The shallow semantics (defined in [BL11] and adopted in our BMC encoding) bears many similarities with the "local-time" semantics defined in [BJLY98] for networks of timed systems and can in fact be seen as a generalization to the hybrid case of [BJLY98]. Indeed, neither requires the synchronization of timed transitions of different components; they both use local clocks that are re-synchronized upon shared events. The two semantics differ in the types of runs used to solve the reachability problem: the shallow semantics consists of sets of local runs, while the local-time semantics consists of runs in the interleaving composition. As far as we know, this is the first attempt to exploit the shallow/local-time semantics to improve BMC.

Partial-Order Reduction (POR) [God96] is one of the most known and used technique to tackle the state-space explosion problem due to interleaving of concurrent systems. The idea is to identify cases when the order of transitions is not relevant in order to prune the search space. The application of POR techniques is difficult in the context of timed and hybrid systems because the timed transitions are global actions which typically interleave the local transition, and thus forbid the pruning performed by POR. The local-time semantics was proposed in [BJLY98] to enable POR by removing the synchronization on timed transitions. The main difference between POR and our encoding is that while POR tackles the interleaving explosion problem by fixing the order of independent transitions, we allow them to be executed in parallel.

Also related is the “step” semantics, used in [HN03] for an efficient encoding of the reachability problem in a network of asynchronous systems. The work in [HN03] is limited to the case of discrete transitions. Our approach can be seen as a generalization of the step semantics to the case of timed transitions.

6.3 K-induction and Implicit Predicate Abstraction

Note. This section presents a joint work with Alessandro Cimatti, Hendro Hendro and Stefano Tonetta.

In this section we use the transition system definition without input variables and the invariant formula.

In the following section we present a verification algorithm based on k-induction and predicate abstraction. The algorithm can be applied to finite and infinite-state transition systems, and thus can be applied also to the infinite-state transition system either obtained from a hybrid automaton H or from a network $\mathcal{N} = H_1 || \ldots || H_n$ of hybrid automata.

The algorithm is suitable to verify infinite-state transition systems. While k-induction can be directly applied to this kind of systems, it may be ineffective. The intuition is that k-induction relies on an equivalence relation to encode the “simple path condition” that avoids “loops” in the symbolic exploration of the transition system paths. If the equivalence relation is the equality between states of the system, then it may be ineffective in the case of infinite-state systems. For example, suppose to have a system with a real counter that is incremented by 1 in each iteration. Then, the simple path condition that uses equalities will not be effective (e.g. the encoding is such that it can always find a new state of the system).

A viable solution is to analyze an abstraction [CGL94] of the original system. Since the abstraction over-approximate the behavior of the system, if there are no paths in the abstraction that violates the invariant property P, then P is satisfied also in the concrete system (i.e. the original system without abstraction). Among the existing abstraction techniques, *Predicate Abstraction* [GS97] abstracts an infinite-state system in a finite-state system: given a set of predicates P, the technique builds a finite-state system where each finite-state represent all the states of the original system that satisfy the same set of predicates. One of the main shortcomings of predicate abstraction is that it needs to existentially quantify several variables (all the concrete variables of the system) both in the initial and in the transition relation of the concrete system. The abstraction can be obtained computing the set of all the satisfying assignment over the set of predicates [LNO06, CFG^{+}10, CDJR09, CCF^{+}07b]. This amounts to perform an $ALLSAT$ enumeration, which can be a bottleneck.

Implicit abstraction [Ton09] is an abstraction technique that perform the model checking task in the abstract space defined by a set of predicates, as in predicate abstraction, but without the need to compute the complete abstraction beforehand. Instead, the abstraction is directly encoded in the unrolling of the infinite-state system, avoiding expensive quantifier elimination procedures. The drawback of the approach is that the number of variables of the system is doubled and there are additional constraints to encode the abstraction. However, the approach is usually more effective than computing the abstraction and model check it, also because one may need to refine the abstraction several times before proving the property.

Instead, the refinement of implicit abstraction may be performed incrementally, supposing that the abstraction is monotonic (i.e. the refinement only adds new predicates, without removing them). We explore refinement techniques based on interpolation [HJMM04], showing that implicit abstraction allow us to minimize the number of predicates to be added during each refinement step. This seems to be really useful when implicit abstraction is combined with k-induction, since the number of predicates may impact on the depth needed to prove the property.

6.3.1 Predicate abstraction

In the following we represent the abstraction by a formula $A(V, \widehat{V})$ such that $s, \widehat{s} \models A$ iff $\widehat{s}$ is the abstraction of s.

Definition 24 (Abstraction) *Given a transition system $S = \langle V, Init, Trans \rangle$, a set $\widehat{V}$ of abstract variables and the relation A, the abstract transition system $\widehat{S} = \langle \widehat{V}_A, \widehat{Init}_A, \widehat{Trans}_A \rangle$ is defined as follows:*

- $\widehat{Init}_A(\widehat{V}) := \exists V(Init(V), \wedge A(V, \widehat{V}))$,
- $\widehat{Trans}_A(\widehat{V}, \widehat{V}') := \exists V, \exists V', (Trans(V, V') \wedge A(V, \widehat{V}) \wedge A(V', \widehat{V}'))$.

Once the set $\widehat{V}$ of abstract variables and the relation A are given, the abstraction $\widehat{S}_A$ is obtained by existentially quantifying the variables of S, V and V'.

The abstraction $\widehat{S}$ preserves the satisfaction of several properties, such as invariants. If $\widehat{S}$ is an abstraction of S, if a condition is reachable in S, then also its abstract version is reachable in $\widehat{S}$. Thus, if $P(V)$ is an invariant property and $\widehat{S} \models \widehat{P}(\widehat{V})$, then $S \models P$.

Predicate abstraction is defined by a set of predicates $\mathbb{P}$, such that each predicate $p \in \mathbb{P}$ is a formula over the variables V of S. We define the set $V_{\mathbb{P}} := \{x_p | p \in \mathbb{P}\}$ of abstract variables (i.e. we define a Boolean variable x_p for each predicate of the abstraction). Then, the abstract relation for predicate abstraction is defined as:

$$A_{\mathbb{P}}(V, V_{\mathbb{P}}) := \bigwedge_{p \in \mathbb{P}} x_p \leftrightarrow p(V) \tag{6.1}$$

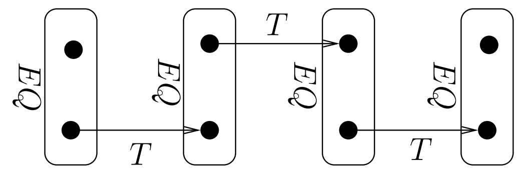


Figure 6.2: Abstract path [Ton09].

Given a formula $\phi(V)$, the abstract version $\widehat{\phi}_{\mathbb{P}}$ is obtained by existentially quantifying the variables V, i.e., $\widehat{\phi}_{\mathbb{P}} = \exists V, (\phi(V) \wedge A_{\mathbb{P}}(V, V_{\mathbb{P}}))$. Similarly for a formula over V and V', $\widehat{\phi}_{\mathbb{P}} = \exists V, V'.(\phi(V, V') \wedge A_{\mathbb{P}}(V, V_{\mathbb{P}}) \wedge A_{\mathbb{P}}(V', V'_{\mathbb{P}}))$.

The abstract system with $\widehat{S}_{\mathbb{P}} = \langle V_{\mathbb{P}}, \widehat{I}_{\mathbb{P}}, \widehat{T}_{\mathbb{P}} \rangle$ is obtained by abstracting the initial and the transition conditions.

Implicit predicate abstraction

Implicit predicate Abstraction [Ton09] (IA) embeds the definition of the predicate abstraction in the symbolic encoding of paths. This is based on the following formula:

$$EQ_{\mathbb{P}}(V, \overline{V}) := \bigwedge_{p \in \mathbb{P}} p(V) \leftrightarrow p(\overline{V})$$

which relate two concrete states corresponding to the same abstract state.

The formula:

$$\widehat{\pi}_{k,\mathbb{P}} := \bigwedge_{1 \leq h < k} (Trans(\overline{V}^{h-1}, V^h) \wedge EQ_{\mathbb{P}}(V^h, \overline{V}^h)) \wedge Trans(\overline{V}^{k-1}, V^k)$$

is satisfiable iff there exists an uninitialized path of k steps in the abstract state space. Intuitively, instead of having a contiguous sequence of transitions, the encoding represents a sequence of disconnected transitions where every gap between two transitions is forced to lay in the same abstract state (see Fig. 6.2). In the following, we assume that

$(\exists V, \neg P(V) \wedge A_{\mathbb{P}}(V, V_{\mathbb{P}})) \leftrightarrow \neg(\exists V, P(V) \wedge A_{\mathbb{P}}(V, V_{\mathbb{P}}))$: this can be guaranteed adding all the predicates of P to the set of predicates $\mathbb{P}$. $BMC_{\mathbb{P}}(k)$ encodes the abstract bounded model checking problem and is obtained from $\widehat{\pi}_{k,\mathbb{P}}$ by adding the abstract initial and target conditions:

$$BMC_{\mathbb{P}}(k) := Init(V^0) \wedge EQ_{\mathbb{P}}(V^0, \overline{V}^0) \wedge \widehat{\pi}_{k,\mathbb{P}} \wedge EQ_{\mathbb{P}}(V^k, \overline{V}^k) \wedge \neg P(\overline{V}^k)$$

Similarly, we obtain the abstract encoding of the forward and backward condition used in k-induction:

$$\begin{aligned}
kindfw_{\mathbb{P}}(k) :=& Init(V^0) \wedge EQ_{\mathbb{P}}(V^0, \overline{V}^0) \wedge simple_{\mathbb{P}}(k) \\
kindbw_{\mathbb{P}}(k) :=& EQ_{\mathbb{P}}(V^0, \overline{V}^0) \wedge simple_{\mathbb{P}}(k) \wedge EQ_{\mathbb{P}}(V^k, \overline{V}^k) \wedge \neg P(\overline{V}^k) \\
simple_{\mathbb{P}}(k) :=& \widehat{\pi}_{k,\mathbb{P}} \wedge \bigwedge_{0 \leq i < j \leq k} \neg EQ_{\mathbb{P}}(V^i, V^j)
\end{aligned}$$

6.3.2 Refinement of implicit predicate abstraction

When the abstract model checking process finds that $\widehat{S}_{\mathbb{P}} \not\models \widehat{P}_{\mathbb{P}}$, we cannot conclude that $S \models P$. Intuitively, since $\widehat{S}_{\mathbb{P}}$ has more paths than S, the abstract counterexample $\widehat{\pi} = \widehat{s}_0; \ldots; \widehat{s}_k$ may be either a spurious (if it is the result of the abstraction) or a real counterexample of S. In the case $\widehat{\pi}$ is spurious, the precision of the abstraction must be increased and the model checking task must be repeated until we obtain a result (i.e. $S \models P$ or $S \not\models P$). We refer to this iterative process as abstraction-refinement loop.

The abstraction-refinement loop of implicit abstraction differs from the standard Counterexample-Guided Abstraction Refinement [CGJ+00] (CEGAR) loop, since the computation and the model checking phase are not separated. Figure 6.3 shows the standard CEGAR loop, which is composed of 4 different phases: (i) computation of the abstraction, (ii) model checking, (iii) simulation of the abstract counterexample, (iv) refinement of the abstraction. Instead, implicit predicate abstraction is composed of

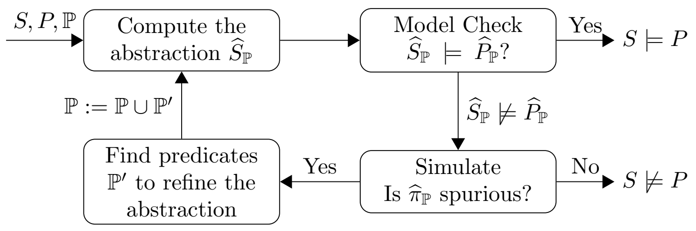

$S, P, \mathbb{P}$
Compute the abstraction $\widehat{S}_{\mathbb{P}}$
Model Check $\widehat{S}_{\mathbb{P}} \models \widehat{P}_{\mathbb{P}}$?
Yes
$S \models P$
$\mathbb{P} := \mathbb{P} \cup \mathbb{P}'$
$\widehat{S}_{\mathbb{P}} \not\models \widehat{P}_{\mathbb{P}}$
Find predicates $\mathbb{P}'$ to refine the abstraction
Yes
Simulate Is $\widehat{\pi}_{\mathbb{P}}$ spurious?
No
$S \not\models P$

Figure 6.3: CEGAR loop.

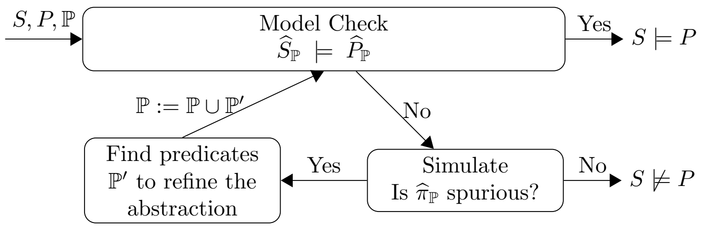

$S, P, \mathbb{P}$
Model Check $\widehat{S}_{\mathbb{P}} \models \widehat{P}_{\mathbb{P}}$
Yes
$S \models P$
$\mathbb{P} := \mathbb{P} \cup \mathbb{P}'$
No
Find predicates $\mathbb{P}'$ to refine the abstraction
Yes
Simulate Is $\widehat{\pi}_{\mathbb{P}}$ spurious?
No
$S \not\models P$

Figure 6.4: IA loop.

3 phases, since the computation of the abstraction and the model checking are not separated (see Figure 6.4). In the following, we focus on the simulation and the refinement phases of the implicit abstraction loop.

A standard technique to simulate an abstract counterexample is BMC. One may perform BMC on the concrete system S up to the length of $\widehat{\pi}$, asserting for each i-th step the assignment to the predicates in the i-th state of $\widehat{\pi}$ (i.e. $\widehat{s_i}$):

$$BMC_{\widehat{\pi}}(k) := BMC_{\mathbb{P}}(k) \wedge \bigwedge_{0 \leq k} (EQ_c(V^i, \overline{V}^i)) \wedge \bigwedge_{0 \leq i \leq k} (\widehat{s_i})$$

where $EQ_c(V, \overline{V})$ forces the encoding of the concrete system S, instead of the predicate abstraction.

$$EQ_c(V, \overline{V}) := \bigwedge_{v \in V} v = \overline{v}$$

If the encoding is satisfiable, then we found a concrete counterexample to P in S, otherwise the counterexample is spurious. For implicit abstraction this procedure may be performed incrementally on the encoding of

$BMC_{\mathbb{P}}(k)$:

If $BMC_{\widehat{\pi}}$ is unsatisfiable, then it means that there are no concrete counterexamples in S that corresponds to $\widehat{\pi}$. In this case, the abstraction can be refined using interpolation [HJMM04]. In the following, we define the prefix and the suffix of $BMC_{\widehat{\pi}}$ as follows:

$$\begin{aligned}
\mathit{Pref}_{\widehat{\pi}}(j) :=& \mathit{Pref}(j) \wedge \bigwedge_{0 \leq i \leq j} (\widehat{s}_i) \\
\mathit{Pref}(j) :=& \mathit{Init}(V^0) \wedge EQ_{\mathbb{P}}(V^0, \overline{V}^0) \wedge \widehat{\pi}_{j,\mathbb{P}} \\
\mathit{Suff}_{\widehat{\pi}}(k-j) :=& \mathit{Suff}(k-j) \wedge \bigwedge_{j \leq i \leq k} (\widehat{s}_i) \\
\mathit{Suff}(k-j) :=& \bigwedge_{j \leq h < k} (Trans(\overline{V}^{h-1}, V^h) \wedge EQ_{\mathbb{P}}(V^h, \overline{V}^h)) \wedge \\
& Trans(\overline{V}^{j-1}, V^j) \wedge EQ_{\mathbb{P}}(V^j, \overline{V}^j) \wedge \neg P(\overline{V}^j)
\end{aligned}$$

where *Pref* and *Suff* are similar to the definition of prefix and suffix of Interpolation-based model checking of Section 2.3.1. For each step $0 \leq i \leq k$ we can compute the interpolant I_i of the formulas $\mathit{Pref}_{\widehat{\pi}}(i)$ and $\mathit{Suff}_{\widehat{\pi}}(k-j)$ such that $I_i \models \mathit{Pref}_{\widehat{\pi}}(i)$ and $\not\models I_i \wedge \mathit{Suff}_{\widehat{\pi}}(k-i)$. Each I_i is defined over the set of variables V^i. Let $I_i[V/\ V^i]$ denote the formula obtained from I_i replacing all the variables $v^i \in V^i$ with $v \in V$ and $\rho(\phi)$ be a function that returns the set of all the predicates of the formula ϕ (e.g. $\rho(x \geq 1 \wedge \neg(y + x = 0))$ returns $\{x \geq 1, x + y = 0\}$).

We refine the abstraction adding the set of predicates $\mathbb{P}' := \bigcup_{0 \leq i \leq k} \rho(I_i)$ to $\mathbb{P}$. As proved in [HJMM04], the predicates will rule out the counterexample $\widehat{\pi}$ from the abstraction.

In practice, the refinement is performed incrementally in the existing encoding of implicit abstraction, asserting the formula $\bigwedge_{0 \leq i \leq k} EQ_{\mathbb{P}'}(V^h, \overline{V}^h)$.

Note that, the predicate abstraction obtained from the set of predicates $\mathbb{P}$ induce an abstract, finite-state model with $2^{\mathbb{P}}$ states. Thus, the number of predicates may have a negative consequences on the performance of the k-induction algorithm, that may be forced to explore more states to prove that a property holds.

The encoding of implicit abstraction allow us to reduce the number of predicates in $\mathbb{P}'$ to obtain a coarser abstraction, exploiting incrementality and the unsatisfiability core extraction.

The formula $BMC_{\mathbb{P}}(k) \wedge \bigwedge_{0\leq i\leq k} EQ_{\mathbb{P}'}(V^h, \overline{V}^h)$ is unsatisfiable, since $\mathbb{P} \cup \mathbb{P}'$ is guaranteed to rule out all the spurious counterexamples at depth k. We define a set of additional Boolean variables $L_{\mathbb{P}'} = \{l_p | p \in \mathbb{P}'\}$ and we consider the formula:

$$BMC_{\mathbb{P}}(k) \wedge \bigwedge_{0\leq i\leq k} \bigwedge_{p\in\mathbb{P}'} (l_p \rightarrow (p(V^i) \leftrightarrow p(\overline{V}^i))) \tag{6.2}$$

The encoding allow us to find a subset of $\mathbb{P}'$ such that the encoding is still unsatisfiable. In practice, we can check if the spurious counterexample is reachable in different abstractions changing the truth values of the Boolean variables in $L_{\mathbb{P}'}$. We use an heuristic algorithm to reduce the predicates in the refinement, using the incremental interface of the SMT solver. Let $\mathbb{P}'' = \emptyset$. First we assert [2] in the solver the formula 6.2: $\gamma(0) := BMC_{\mathbb{P}}(k) \wedge \bigwedge_{0\leq i\leq k} \bigwedge_{p\in\mathbb{P}'} (l_p \rightarrow (p(V^h) \leftrightarrow p(\overline{V}^h)))$ Then, we assert a Boolean variable $l_p \in L_{\mathbb{P}'}$, $\mathbb{P}'' = \mathbb{P}'' \cup \{p\}$, and we check the satisfiability of the encoding. If the encoding is satisfiable, then we chose another predicate p' from $\mathbb{P}' \setminus \mathbb{P}''$, we add it to $\mathbb{P}''$ and we assert l'_p it in the solver. Eventually the encoding will be unsatisfiable. The set $\mathbb{P}'' \subseteq \mathbb{P}'$ is the set of reduced predicates. Note that, in the worst case $\mathbb{P}'' = \mathbb{P}$. In practice, we reduce the size of $\mathbb{P}''$ keeping only its elements that appear in the unsatisfiable core extracted by the solver.

Note that we do not obtain neither the minimum nor a minimal set of predicates. An algorithm that computes a minimal set of elements is presented in [BM07b].

[2]In the real implementation the formula $BMC_{\mathbb{P}}(k)$ is already in the solver stack.

6.3.3 Related Work

The reachability problem for hybrid systems, and network of hybrid systems, has been extensively studied in the literature (see [Alu11] for a recent survey). A possible classification of the existing approaches to prove invariant properties divides them in three categories: symbolic reachability [HHWT97, FGD$^+$11b, TK04, BBC$^+$12, BBC$^+$14, RS07], deductive verification [Pla08, PJ04], and abstraction-based approaches [ADI06, CFH$^+$03, Tiw08].

The symbolic transition system encoding presented in Part II allow us to reduce the reachability problem of hybrid automata to the reachability problem of symbolic transition systems. The main contribution of K-induction and implicit predicate abstraction is orthogonal to the verification technique for hybrid systems, since the technique works for general infinite-state transition systems.

The main difference of implicit predicate abstraction [Ton09] with respect to the the standard abstraction techniques, such as [LNO06, CFG$^+$10, CDJR09, CCF$^+$07b], is that the abstraction has not to be computed before starting the model checking process. Note that, for predicate abstraction, the computation of the abstraction is the main bottleneck of the verification task. A common way to tackle the complexity of predicate abstraction is to approximate the computation by allowing more transitions (see, e.g., [CGJ$^+$00, BPR03, STT09]). The complexity is shifted to the refinement that must take care of removing spurious transitions, resulting in an increased number of refinement iterations. Instead, we always consider the abstraction induced by the set of predicates.

There exists a wide literature of technique that can be applied to verify infinite-state transition systems.

Several approaches apply k-induction on the concrete model [dMRS03, Pik07, SDS08, SD10]. However, the main issue with the approach is that the inductive step may be ineffective on infinite-state systems, where there exist infinite-paths that are not lasso-shaped.

Interpolation-based model checking [McM03] may be applied on the infinite-state transition system. The algorithm works on the concrete system but, since it builds an over-approximation of the set of the reachable states, it may be effective.

Recently, several approaches adapted the original IC3 algorithm [Bra11] to deal with infinite-state systems [CG12, HB12, KJN12b, WK13]. The techniques presented in [CG12, HB12] extend IC3 to verify systems described in the linear real arithmetic theory, and thus can be applied to the symbolic transition system encodings of linear hybrid automata. These techniques perform a search in the concrete space, but they may be effective since they incrementally construct an inductive invariant to prove the property.

In a recent work [CGMT14a] we applied the implicit abstraction framework to IC3. The result is an efficient algorithm that is not sensitive to the number of predicates, and that extends the use of IC3 to transition systems encoded in any background theory, provided that the satisfiability of the theory is decidable and there is an effective refinement method (e.g. interpolation). The main difference is that this work uses IC3 as backend algorithm.

Chapter 7

Scenario verification

Note. The material presented in this chapter has already been presented in [CMT11a, CMT11c, CMT13c].

In this chapter, we concentrate on the problem of scenario-based verification, that consist of checking if a network of hybrid automata accepts some desired interactions among the components.

We use a variant of Message Sequence Charts [IT96] (MSCs) extended with constraints, to express scenarios of such interactions. MSCs are especially useful for the end users because of their clarity and graphical content. The ability to check whether a MSC is feasible in a network of HAs (i.e. if the network may exhibit behaviors that satisfy a given MSC) is an important feature to support user validation.

In principle, the problem could be reduced to the reachability of an accepting state in the composition of the network encoding and an observer automaton, which is obtained translating the MSC. However, all the different construction may be inefficient and may not scale to more complex models and scenarios. This because the verification algorithms must explore the paths in the composition of the system and the monitor. The length of these paths depends on the model and also on the length of the MSC.

We propose a novel approach that exploits the structure of the scenario to partition and drive the search. We propose two different algorithms based on Bounded Model Checking and k-induction, to either demonstrate that a MSC is feasible or unfeasible for a network of hybrid automata. Both the algorithms are based on an encoding similar to “shallow synchronization” (See Section 6.2.1), where the encoding of each automata is unrolled independently, and the synchronizations among the components may happen at different steps.

In the case of BMC, we propose an encoding that is structured around the events in the MSC, which are used as intermediate “islands”. The idea is to pre-simplify fragments of the encoding based on the events attached to the islands. The algorithm proceeds by incrementally increasing the length of the local paths between two consecutive islands and linking the local path to the next island by means of equalities. The k-induction algorithm, that is used to prove the MSCs unfeasibility, is specialized to the structure of the MSC, so that the “simple path” condition is localized in the fragments between the events, rather than being imposed globally on the whole network.

The encoding is also suitable to generate additional information for the end user in the case a scenario is unfeasible. The problem of generating an explanation for the unfeasibility of an MSC is of practical interest. Opposed to the feasible case, where the algorithm may produce a simulation trace, in the unfeasible case the user does not have any additional debug information. We also provide techniques to obtain additional information that explain the reasons for unfeasibility (e.g. which components are involved, which temporal restrictions between events in the MSC are too strong).

The techniques exploit the SMT solver capabilities, like incrementality, unsatisfiable core extraction and interpolation to generate explanations.

First, in Section 7.1 we define the scenario-verification problem. Then, we present the specialized verification techniques for both feasibility (Section 7.2) and unfeasibility (Section 7.3). In Section 7.4 we describe different explanation of unfeasibility for an MSC, and their usefulness on several case studies. We conclude the Chapter discussing related works.

7.1 Problem definition

In order to support user validation, it is very important to be able to check whether a HAN may exhibit behaviors that satisfy a certain scenario, specifying some desired or undesired interactions among the components. We use the language of Message Sequence Charts (MSCs) [IT96] and its extensions to express scenarios of such interactions.

An MSC defines a single (partial-order) interaction of the components of a network $\mathcal{N} = H_1||\ldots||H_n$. MSCs have been extended in several ways. We consider here a particular variant, enriched with additional constraints, which turns out to be very useful and easy to handle with the SMT-based approach.

An MSC m is associated with a set of events $A_m \subseteq A_{\mathcal{N}}$, where $A_{\mathcal{N}} = \bigcup_{1\leq i\leq n} A_i$ is the set of all the events of the network $\mathcal{N}$. The typical implicit assumption is that the set A_m contains all the synchronization events of the network. Since we are dealing with networks of hybrid automata the timed event is not part of A_m and, thus, is not present in the sequence of events specified by the MSC. Therefore, we assume that if $\mathcal{N}$ is a network of the hybrid automata $H_1, \ldots, H_n$ with alphabet respectively $A_1, \ldots A_n$, then $A_m = \bigcup_{1\leq i<j\leq n} A_i \cap A_j$ and thus the timed event is not part of A_m[1].

The MSC defines a sequence of events for every component H_i of the net-

[1]The techniques presented in this chapter can be adapted to handle synchronizations which are not in A_m.

work, called *instance* of H_i. An instance σ_i of H_i is a sequence $a_1; \ldots; a_l \in (A_m \cap A_i)^*$ of events of H_i. We denote the j-th event a_j of the instance σ_i with $\sigma_i[j]$ and the length of σ_i with $|\sigma_i|$. Along each instance line σ_i there is a finite set of *local segments* $\{lsg(\sigma_i[0]), \ldots, lsg(\sigma_i[l])\}$ that denote the position between two consecutive events: $lsg(\sigma_i[j])$ is the local segment between the j-th event and the $j+1$-th event of σ_i. The first local segment from the beginning of the instance to the first event is $lsg(\sigma_i[0])$ and the final local segment after the $|\sigma_i|$-th event is $lsg(\sigma_i[l])$.

In the following, we associate to each hybrid automaton H_i an infinite-state transition system S_i, defined as in the local time semantic encoding of Section 4.2. Also, we denote with τ_i the set of local events of the i-th automaton in the network, i.e. $\tau_i = A_i \subseteq \bigcup_{j \neq i} A_j$. *$H_i$ accepts the instance σ_i with respect to A_m* iff the FOTS S_i of H_i *accepts σ_i with respect to A_m* ($S_i \models^m \sigma_i$). $S_i \models^m \sigma_i$ iff there exists a trace w accepted by S_i such that the sub-sequence of events in A_m of w is equal to σ_i (i.e. $w_{|\bigvee_{a \in A_m} \varepsilon = a} = \sigma_i$). In this case we say that w is *compatible* with σ_i. In other words, S_i accepts the instance with respect to A_m iff there exists a path π of S_i over a trace compatible with σ_i. In such cases, we write $\pi \models^m \sigma_i$.

If $\pi \models^m \sigma$, π must be in the form $s_0; \varepsilon = \tau; \ldots; \varepsilon = \tau; s_{h_1}; \varepsilon = \sigma[1]; s_{(h_1+1)}; \varepsilon = \tau; \ldots; \varepsilon = \tau; s_{h_{|\sigma|}}; \varepsilon = \sigma[|\sigma|]; s_{(h_{|\sigma|}+1)}; \varepsilon = \tau; \ldots; \varepsilon = \tau; s_{h_{(|\sigma|+1)}}$, where s_h is a model over the state variables V_i of S_i and τ are local events of H_i. We denote the sub-sequences of the path π in which it is split by σ as follows:

- $pre_j(\pi) = s_{h_j}$ is the source state of the transition labeled with $\varepsilon = \sigma[j]$ in π.
- $post_j(\pi) = s_{h_j+1}$ is the destination state of the transition labeled with $\varepsilon = \sigma[j]$ in π.
- $loc_j(\pi) = s_{h_j+1}; \ldots; s_{h_{j+1}}$ is the sequence of states between the j-th and the $j+1$-th shared events, where we denoted 0 with h_0.

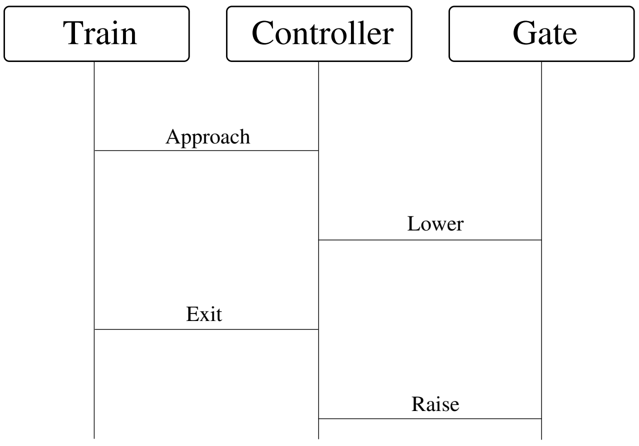


Figure 7.1: An MSC for the Train-Gate-Controller [Hen96].

An MSC is the parallel composition $\sigma_1||\ldots||\sigma_n$ where σ_i is an instance of H_i. An MSC $\sigma_1||\ldots||\sigma_n$ is *consistent* iff for every pair of instances σ_i and σ_j the projection on the common alphabet is the same, i.e., if $A = A_i \cap A_j$, $\sigma_{i|A} = \sigma_{j|A}$. Henceforth, we assume that the MSCs are consistent.

The network $\mathcal{N}$ accepts the MSC m iff the FOTS $S_{\text{LocTime}}(\mathcal{N}) = S_1||\ldots||S_n$ accepts m ($S_{\text{LocTime}}(\mathcal{N}) \models m$). $S_{\text{LocTime}}(\mathcal{N}) \models m$ iff there exists a trace w accepted by $S_{\text{LocTime}}(\mathcal{N})$ such that, for every S_i, the sub-sequence of events in $A_m \cap A_i$ is equal to σ_i ($w_{|(\bigvee_{a\in A_m} \varepsilon=a)} = \sigma_i$). In other words, $S_{\text{LocTime}}(\mathcal{N})$ accepts the MSC m iff there exists a path of $S_{\text{LocTime}}(\mathcal{N})$ over a trace compatible with every instance of the MSC.

$V_m := \bigcup_{1\leq i\leq n} V_{\sigma_i}$ is the set of variables of the CMSC m, where for all $1 \leq i \leq n$, $V_{\sigma_i} := \bigcup_{0\leq j\leq(|\sigma_i|+1)} V_i^j$ (e.g. v_i^j represents the value of the variable v_i of the i-th component just before the j-th event $\sigma_i[j]$ of σ_i). We define a *Constrained MSC (CMSC)* as a pair $\langle m, \phi\rangle$ where m is an MSC $\sigma_1||\ldots||\sigma_n$, $\phi = \phi_0 \wedge \phi_1 \wedge \ldots \wedge \phi_n$, ϕ_0 is a formula over V_m and for all $1 \leq i \leq n$, ϕ_i is a formula over V_i^j. Given a path $\pi = s_0; a_1; s_1; \ldots; a_k; s_k$ of $\mathcal{N}_{\text{LocTime}}(\mathcal{H})$, the *projection* of π over S_i is the path $prj(\pi, i)$ obtained projecting the states over the S_i-th component and removing all the transitions over events which are not in A_i. $S_{\text{LocTime}}(\mathcal{N}) \models \langle m, \phi \rangle$ iff there exists a path π of $S_{\text{LocTime}}(\mathcal{N})$ such that $\bigcup_{1 \leq i \leq n, 0 \leq j \leq |\sigma_i|} pre_j(prj(\pi, i)) \models \phi$.

Example 10 *Figure 7.1 shows an MSC for the railroad model from [Hen96]. There is an instance for each automaton in the network, Train, Controller and Gate. The MSC represents a scenario where the Train communicates with the controller when approaching the Gate and the controller synchronizes with the Gate to close it. When the Train is far, it synchronizes with the Controller, which opens the Gate.*

The model checking problem for a CMSC $\langle m, \phi \rangle$ is the problem of checking if a network satisfies a CMSC.

The classical approach is based on the construction of a monitor that, composed with $S_{\text{LocTime}}(\mathcal{N})$, forces $S_{\text{LocTime}}(\mathcal{N})$ to follow only paths that satisfy the MSC. It is in spirit similar to the automata-approach to LTL model checking [Var95]. The SMT-based verification techniques are applied off the shelf on the resulting FOTS. The monitor can be one additional component in the network or consist of many components one for each instance of the CMSC. Exploiting local-time semantics, the monitor can also be reduced to follow one interleaving of the partial-order reduction defined by the CMSC. However, the experimental analysis of [CMT11a] shows that the best option is to use a monitor per component.

7.2 Scenario-driven BMC

The drawbacks of the traditional SMT-based encoding is that it cannot exploit the sequence of messages prescribed by the MSC in order to simplify the search because of the uncertainty on the number of local steps between two events. We encode the path of each automaton independently, exploiting the local time semantics, and then we add constraints that force shared events to happen at the same time, as in *shallow synchronization* [BCL$^+$10a]. Moreover, we fix the steps corresponding to the shared events and we parametrize the encoding of the local steps with a maximum number of transitions. The encoding is conceived in order to maximize the incrementality of the solver, as described in Section 2.1.3, along the increase of the number of local steps. The idea is that we keep the encodings of the sequences of local transitions separated from the encoding of the next shared event, and we unroll them incrementally, while we add and remove accordingly the equality constraints which glue such sequences. Note that the encoding of a sequence of local transitions does not fix the exact number of local transitions, since we allow the stutter action. This because we do not known a priori what will be the exact length of each local path.

Let us consider a hybrid automata network $\mathcal{N} = H_1 || \dots || H_n$ and the correspondent FOTS $S_i = \langle V_i, \mathcal{W}_i, Init_i, Inv_i, Trans_i \rangle$, representing H_i, for $1 \leq i \leq n$, in the local-time semantics. We denote with $T_{i|\phi}$ the transition condition restricted to the condition ϕ, i.e., $T_{i|\phi} = Trans_i \wedge \phi$. We abbreviate $T_{i|\varepsilon=a}$ with $T_{i|a}$ and $T_{i|\varepsilon \in \tau_i \cup \{\mathrm{S},\mathrm{T}\}}$ with $T_{i|\tau}$. We associate a bound $k_i[j]$ to the j-th segment $lsg(\sigma_i[j])$ of the i-th instance. $k_i[j]$ is used to limit the number of transitions in the local path $loc_j(\pi)$ of a path π satisfying the instance σ_i. We use $\overline{k}_i$ to denote $\langle k_i[0], \dots, k_i[h_{|\sigma_i|}] \rangle$ and $\overline{k}$ to denote $\langle \overline{k}_1, \dots, \overline{k}_n \rangle$. Moreover, for all $1 \leq i \leq n$ and $0 \leq j \leq |\sigma_i|$, we define the index $idx_i[j]$ such that $idx_i[0] = -1$ and if $\langle i, j \rangle \neq \langle i, j' \rangle$ then $idx_i[j] + h \neq idx_i[j'] + h'$ for all h, h' with $0 \leq h \leq k_i[j]$ and $0 \leq h' \leq k_i[j']$ (i.e. the indexes of two different segments do not overlap).

The following encoding represents all paths of the network compatible with the MSC where the local transitions of the j-th segment of the i-th instance have been unrolled up to $k_i[j]$ times (note that the "up to" is due to the ability of stuttering):

$$\begin{array}{rcl}
enc(m,\overline{k}) & := & \bigwedge\limits_{1\leq i\leq n} enc(\sigma_i,\overline{k}_i) \wedge \bigwedge\limits_{1\leq j<i\leq n} sync(\sigma_j,\sigma_i) \wedge \\
& & finalsync(m,\overline{k}) \\
enc(\sigma_i,\overline{k}_i) & := & Init_i^0 \wedge Inv_i^0 \wedge enc(\sigma_i,k_i[0]) \wedge \\
& & \bigwedge\limits_{1\leq j\leq|\sigma_i|} (V^{idx_i[j-1]+k_i[j-1]+1} = V^{idx_i[j]} \wedge \\
& & T_{i|\sigma_i[j]}^{idx_i[j]} \wedge Inv_i^{idx_i[j]} \wedge enc(\sigma_i,k_i[j])) \\
enc(\sigma_i,k_i[j]) & := & \bigwedge\limits_{1\leq h\leq k_i[j]} (T_{i|\tau}^{idx_i[j]+h} \wedge Inv_i^{idx_i[j]+h}) \\
sync(\sigma_j,\sigma_i) & := & \bigwedge\limits_{1\leq z\leq|\sigma_{j|A_{\{i,j\}}}|=|\sigma_{i|A_{\{i,j\}}}|} t_i^{idx_i[f_i^{ij}(z)]} = t_j^{idx_j[f_j^{ij}(z)]} \\
finalsync(m,\overline{k}) & := & \bigwedge\limits_{1\leq i<n,j=i+1} (t_i^{idx_i[|\sigma_i|]+k_i[|\sigma_i|]+1} = t_j^{idx_j[|\sigma_j|]+k_j[|\sigma_j|]+1})
\end{array}$$

where $A_{\{i,j\}} = A_i \cap A_j$ and the function f_i^{ij} maps the z-th event a_z shared between σ_i and σ_j to the index of a_z in σ_i. More, specifically, if $\sigma_{j|_A} = \sigma_{i|_A} = a_1; \ldots; a_l$, then $f_i^{ij}, f_j^{ij} : \mathbb{N} \to \mathbb{N}$ are such that $a_z = \sigma_i[f_i^{ij}(z)] = \sigma_j[f_j^{ij}(z)]$, for $1 \leq z \leq l$.

Intuitively, $enc(m,\overline{k})$ encodes the unrolling of each component according to its instance and guarantees that the different unrollings have the same time for every occurrence of a shared event and the same final time. In order to encode the paths that satisfy a CMSC $\langle m,\phi\rangle$ we have just to conjoin the additional constraints ϕ:

$$enc(\langle m,\phi\rangle,\overline{k}) \;:=\; enc(m,\overline{k}) \wedge \phi[v_i^{idx_i[j]}/v_i[j]]$$

where for all the instances i, $1 \leq i \leq n$, and all events j, $1 \leq j \leq |\sigma_i|$, we substitute $v_i[j]$ in ϕ with the timed variable $v_i^{idx_i[j]}$.

Example 11 *The Figure 7.2 shows the encoding of the MSC of Example 10, fixing the length of all the local steps to* 1 *(we abbreviated the name of the components with their initial letter). Each sequence of states represents the encoding for a single instance of the MSC, the double arrows represent the links of the local segments with the shared events, and the dotted lines represent the equalities over the local time variables.*

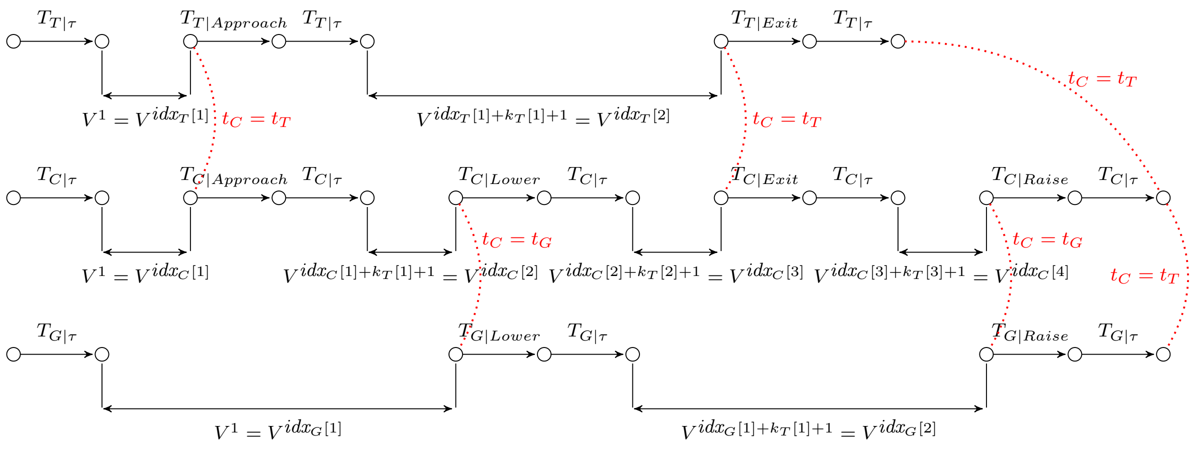


Figure 7.2: Encoding for the Train-Gate-Controller MSC fixing to 1 the length of all the local bounds.

Theorem 11 *If* $enc(\langle m, \phi\rangle, \overline{k})$ *is satisfiable then* $\mathcal{N} \models \langle m, \phi\rangle$*. Vice versa, if* $\mathcal{N} \models \langle m, \phi\rangle$*, then there exist integers* $\overline{k}$ *such that* $enc(\langle m, \phi\rangle, \overline{k})$ *is satisfiable.*

Proof. Let us consider a model μ of the formula $enc(\langle m, \phi\rangle, \overline{k})$. μ is a model over the variables of the network $S_{\text{LocTime}}(\mathcal{N})$. For all $1 \leq i \leq n$, consider the projection π_i of μ over the symbols defined in the Σ-structure V_i and $\mathcal{W}_i$ of S_i. By construction π_i is a path of S_i (i.e. $\pi_i \models S_i$), $\pi_i \models^m \sigma_i$ and $\bigcup_{1\leq i\leq n, 0\leq j\leq|\sigma_i|} pre_j(\pi_i) \models \phi$. Moreover, for all $1 \leq i < j \leq j$, π_i and π_j are consistent, since the MSC is consistent and the events happen at the same time due to *sync*. Thus, $\langle \pi_1, \ldots, \pi_n\rangle$ is a shallowly synchronized path (See Definition 22). By Theorem 10, there exists a path π_{LocTime} in $S_{\text{LocTime}}(\mathcal{N})$ such that $\pi_{\text{SHALLOW}} := \langle prj(\pi_{\text{LocTime}}, 1), \ldots, prj(\pi_{\text{LocTime}}, n)\rangle$. By the definition in Section 7.1, it means that $\mathcal{N} \models \langle m, \phi\rangle$.

If $\mathcal{N} \models \langle m, \phi\rangle$, then there is a path π_{LocTime} such that $\pi_{\text{LocTime}} \models S_{\text{LocTime}}(\mathcal{N})$. This means that $prj(\pi_{\text{LocTime}}, i) \models^m \sigma^i$, $prj(\pi_{\text{LocTime}}, i) \models S_i$ and $\bigcup_{1\leq i\leq n, 0\leq j\leq|\sigma_i|} pre_j(prj(\pi_{\text{LocTime}}, i)) \models \phi$.
By Theorem 10, $\langle prj(\pi_{\text{LocTime}}, 1); \ldots; prj(\pi_{\text{LocTime}}, n)\rangle$ forms a shallowly synchronized path. Let us consider a $\overline{k}$ such that, for all $1 \leq i \leq n$, for all $0 \leq j \leq |\sigma_i|$, $k_i[j]$ is equal to the length of the local segment $lsg(\sigma_i[j])$ in $prj(\pi_{\text{LocTime}}, i)$. $prj(\pi_{\text{LocTime}}, 1), \ldots, prj(\pi_{\text{LocTime}}, n) \models enc(\langle m, \phi\rangle, \overline{k})$, since each $prj(\pi_{\text{LocTime}}, i) \models enc(\sigma_i, \overline{k}_i)$, the projections of the paths satisfy the constraints of the CMSC ϕ by definition (i.e. for each automaton i, $1 \leq i \leq n$, and for each segment of the CMSC j, $0 \leq j \leq |\sigma_i|$, $pre_j(prj(\pi_{\text{LocTime}}, i)) \models \phi$), and, since $\langle prj(\pi_{\text{LocTime}}, 1); \ldots; prj(\pi_{\text{LocTime}}, n)\rangle$ is a shallowly synchronized path, the *sync* constraints holds in $enc(\langle m, \phi\rangle, \overline{k})$ (i.e. the synchronization happen at the same time and the time at the end of all the paths is the same). $\diamond$

In the following, we detail how we increase the bound of the local transitions incrementally adding new constraints to the solver. We consider the same number k of local steps for each segment of the CMSC. This simplifies the algorithm because it is not necessary to decide which segment to increment and reduces the number of calls to the SMT solver.

With regard to the formulas introduced in Section 2.1.3 to describe the incremental interface to the SMT solver[2], we define the partial encoding

[2]In the definition the γ-s formulas are asserted in the solver and never removed, while the β-s formulas are asserted and removed in every satisfiability check.

for an instance σ_i as follows:

$$\begin{aligned}
\gamma_{enc(\sigma_i)}(0) &:= Init_i^0 \wedge Inv_i^0 \wedge \bigwedge_{1\leq j\leq|\sigma_i|} T_{i|\sigma_i[j]}^{idx_i[j]} \wedge Inv_i^{idx_i[j]} \\
\gamma_{enc(\sigma_i)}(k) &:= \bigwedge_{0\leq j\leq|\sigma_i|} T_{i|\tau}^{idx_i[j]+k} Inv_i^{idx_i[j]+k} \\
\beta_{enc(\sigma_i)}(k) &:= \bigwedge_{0\leq j<|\sigma_i|} V^{idx_i[j]+k+1} = V^{idx_i[j+1]}
\end{aligned}$$

For each instance σ_i we encode the initial condition and all the $|\sigma_i|$ events in $\gamma_{enc(\sigma_i)}(0)$. We incrementally increase the length of the local step in $\gamma_{enc(\sigma_i)}(k)$ and in $\beta_{enc(\sigma_i)}(k)$, which glues the last state of a sequence of local steps with the first state that performs the next shared event.

The incremental encoding considering the whole MSC m is defined as follows:

$$\begin{aligned}
\gamma(0) &:= \bigwedge_{1\leq i\leq n} \gamma_{enc(\sigma_i)}(0) \wedge \bigwedge_{1\leq i<j\leq n} sync(\sigma_i, \sigma_j) \\
\gamma(k) &:= \bigwedge_{1\leq i\leq n} \gamma_{enc(\sigma_i)}(k) \\
\beta(k) &:= \bigwedge_{1\leq i\leq n} \beta_{enc(\sigma_i)}(k) \wedge finalsync(m, \overline{k})
\end{aligned}$$

7.2.1 Invariant generation

In order to strengthen the scenario-driven encoding, and thus speed up the search, we generate invariants for each automata in the network.

For each automaton H_i of $\mathcal{N}$, we compute a finite-state abstraction of its symbolic transition system S_i, Then, we use standard techniques, in our case BDDs, to compute invariants which holds in different sections of σ_i. In particular, we find an over-approximation of the set of the reachable states of $\hat{S}_i$ between two shared events, just before an event, and just after an event. The invariants are then conjoined to the scenario encoding.

Given an MSC instance $\sigma_i = a_1; \ldots; a_h$ for the system S_i, for all $0 \leq u \leq h$, we find the constraints $pre_u, post_u$, and loc_{ui}, such that, for every π such that $\pi \models \sigma$:

- for all u, $0 \leq u \leq h$, $pre_j(\pi) \models loc_u$;
- for all u, $1 \leq u \leq h$, $post_j(\pi) \models pre_u$;
- for all u, $1 \leq u \leq h$, $loc_j(\pi) \models post_u$.

Thus, the constraints over-approximate the set of the reachable states of the network that are consistent with the MSC. We can safely strengthen the encoding of the scenario with such constraints in order to speed up the search.

We perform the invariant generation process for S_i and σ in three different steps: we compute the abstraction $\hat{S}_i$, we perform a forward reachability analysis computing a first set of invariants and finally we refine the invariants with a backward reachability analysis. We compute the Boolean Abstraction $\hat{S}_i$ of S_i, replacing each predicate of S_i with a fresh Boolean variable, and we represent $\hat{S}_i$ with Binary Decision Diagrams (BDDs). Then, we perform a forward reachability analysis on $\hat{S}_i$ computing an over-approximation of $post_i$ and loc_i. We start the reachability analysis from the initial states of $\hat{S}_i$, and we compute loc_0 with a fixed-point of the image restricted to the local events τ. Then, starting from loc_0, we compute $post_1$ with a single image computation restricted to a_1. We alternate these two steps for all a_i of σ. Finally, we perform a backward reachability analysis on $\hat{S}_i$ to compute pre_i and to refine $post_i$ and loc_i. We start from $post_h$ and we compute the precise pre_h as the intersection of loc_{h-1} and the pre-image of $post_h$ restricted to the event a_h. Then, we refine loc_{h-1}, intersecting it with the fixed-point of the pre-image which starts from pre_h and is restricted to τ. At this point we refine $post_{h-i}$, intersecting it with loc_{h-1}. We iterate these steps following σ in reverse order.

7.3 Scenario-driven Induction

In this section, we describe how the structure of the MSC can be exploited to tailor k-induction to prove the unfeasibility of the scenario. For the base case, we use the encoding of Section 7.2. For the inductive step, we apply the simple path condition to each segment of the scenario. The use of different local bounds as presented in Section 7.2 allows k-induction to stop the unrolling of the local path at different depths according to the local structure of the component at the considered segment.

The goal is to find an inductive condition $kind(\langle m, \phi\rangle, \overline{k})$ such that, if $kind(\langle m, \phi\rangle, \overline{k})$ and $enc(\langle m, \phi\rangle, \overline{k})$ are unsatisfiable for some $\overline{k}$, then $\mathcal{N} \not\models \langle m, \phi\rangle$. It is not possible to apply the simple path condition independently to each segment of the encoding. There exists two main difficulties. The first is that the projection of a simple path on a component may not be a simple path. The second is that if a simple path is the concatenation or the parallel composition of two paths, these may not be the longest simple paths of their segments.

The CMSC $\langle m, \phi\rangle$ defines a partial order $\leq_m$ among the segments of m defined as the transitive closure of the smallest relation such that:

- $lsg(\sigma_i[j]) \leq_m lsg(\sigma_i[j'])$ if $0 \leq j \leq j' \leq |\sigma_i|$;
- $lsg(\sigma_i[j]) \leq_m lsg(\sigma_{i'}[j'])$ if $lsg(\sigma_i[j]) = lsg(\sigma_{i'}[j'])$ or there exists a $lsg(\sigma_{i''}[j''])$ such that there is a synchronization between $\sigma_i[j]$ and $\sigma_{i''}[j'']$ and $lsg(\sigma_{i''}[j'']) \leq_m lsg(\sigma_{i'}[j'])$.

Given a CMSC $\langle m, \phi\rangle$ and the local segment $lsg(\sigma_i[j])$, $\langle \overline{m}_i[j], \overline{\phi}_i[j]\rangle$ is partial CMSC where:

- $\overline{m}_i[j] = \overline{\sigma}_1 || \ldots || \overline{\sigma}_n$ such that for all $1 \leq v \leq n$, $|\overline{\sigma}_v| \leq |\sigma_v|$ and for all $1 \leq z \leq |\overline{\sigma}_v|$ $\overline{\sigma}_v[z] = \sigma_v[z]$ and $lsg(\overline{\sigma}_v[z]) \leq_m lsg(\sigma_i[j])$ or $lsg(\overline{\sigma}_v[z]) = lsg(\sigma_i[j])$, while for all $|\overline{\sigma}_v| < z \leq |\sigma_v|$ $lsg(\sigma_v[z]) \not\leq_m lsg(\sigma_i[j])$.

- $\overline{\phi}_i[j]$ is the conjunction of all the conjuncts of ϕ which are over variables in $\overline{m}_i[j]$.

We define the local simple path condition as follows:

$$\begin{array}{rcl} kind_i[j] & := & enc(\langle \overline{m}_i[j], \overline{\phi}_i[j] \rangle, \overline{k}) \wedge simple_i[j] \\ simple_i[j] & := & \bigwedge\limits_{1 \leq h \leq z \leq k_i[j]} s_i^{idx_i[j]+h} \neq s_i^{idx_i[j]+z} \end{array}$$

Theorem 12 *If there exist $\overline{k}$ s.t. $enc(\langle m, \phi \rangle, \overline{k})$ is unsatisfiable and, for all i, j, $kind_i[j]$ is unsatisfiable, then $\mathcal{N} \not\models m$.*

To prove the theorem we provide two additional lemmas.

We introduce the following notation short-hands. Given a tuple of bounds $\overline{k}$ we write $\overline{k}' \geq \overline{k}$ iff for all $1 \leq i \leq n$, for all $0 \leq j \leq |\sigma_i|$, $k'_i[j] \geq k_i[j]$. Given a path π such that $\pi \models^m \sigma_i$, $|loc_j(\pi)|$ is the length of the sequence of states $loc_j(\pi)$ of π.

The CMSC $\langle m, \phi \rangle$, defines a partial order $<_m$ among the segments of m defined as the transitive closure of the smallest relation such that:

- $lsg(\sigma_i[j]) <_m lsg(\sigma_i[j'])$ if $0 \leq j \leq j' < |\sigma_i|$;
- $lsg(\sigma_i[j]) <_m lsg(\sigma_{i'}[j'])$ if there exists a $lsg(\sigma_{i''}[j''])$ such that there is a synchronization between $\sigma_i[j]$ and $\sigma_{i''}[j'']$ and $lsg(\sigma_{i''}[j'']) <_m lsg(\sigma_{i'}[j'])$.

Note that $<_m$ is defined in a similarly to $\leq_m$.

Lemma 2 *Given a CMSC $\langle m, \phi \rangle$ and a tuple of bounds $\overline{k}$, if $enc(\langle m, \phi \rangle, \overline{k})$ is satisfiable then for all $\overline{k}' \geq \overline{k}$, $enc(\langle m, \phi \rangle, \overline{k}')$ is satisfiable.*

Lemma 2 states that if $enc(\langle m, \phi \rangle, \overline{k})$ is satisfiable, then all the encodings which consider a $\overline{k}' >= \overline{k}$ are satisfiable as well, due to the insertion of stuttering action.

Lemma 3 *For all* $1 \leq i \leq n$, $0 \leq j \leq |\sigma_i|$, *if* $kind_i[j]$ *is unsatisfiable for bounds* $\overline{k}$, *then for all paths* π *such that* $\pi \models enc(\langle \overline{m}_i[j], \overline{\phi}_i[j] \rangle, \overline{k})$ *and* $|loc_j(prj(i, \pi))| > k_i[j]$, $loc_j(prj(i, \pi))$ *is not simple.*

The proof of both lemma is available in the Appendix A.1

Proof. Theorem 12 Suppose that $\mathcal{N} \models m$.

By Theorem 11 there exist bounds $\overline{k}'$ and a path π' such that:

- $\pi' \models enc(\langle m, \phi \rangle, \overline{k}')$;
- for all $1 \leq i \leq n$, $0 \leq j \leq |\sigma_i|$, $loc_j(prj(\pi', i))$ is of length $k_i'[j]$.

We consider the bounds $\overline{k}'$ such that $\overline{k}' \geq \overline{k}$. If we show that the formula $enc(\langle m, \phi \rangle, \overline{k}')$ is unsatisfiable, then by Lemma 2 we known that for all the $k'' \leq k$ $enc(\langle m, \phi \rangle, \overline{k}'')$ is unsatisfiable.

For all $1 \leq i \leq n$, $0 \leq j \leq |\sigma_i|$, by Lemma 3 it does not exist a π'' consistent with $\langle m, \phi \rangle$ such that its local segment $loc_j(prj(\pi'', i))$ is simple and longer than $k_i[j]$.

Thus, there exists a path π such that:

- for all $1 \leq i \leq n$, $0 \leq j \leq |\sigma_i|$:
  - $pre_j(prj(\pi, i)) = pre_j(prj(\pi', i))$;
  - $post_j(prj(\pi, i)) = post_j(prj(\pi', i))$;
  - the length of loc_j is $k_i[j]$.
- $\pi \models enc(\langle m, \phi \rangle, \overline{k})$.

This contradicts the fact that by hypothesis $enc(\langle m, \phi \rangle)$ is unsatisfiable. $\diamond$

In order to check if k-induction holds incrementally, we visit the MSC m according to the partial order $\leq_m$. We incrementally apply the partitioned simple path condition to the local segments of m. The incremental checks exploit the incremental interface of the solver.

The structure of local transitions between two shared events is often simple and without loops. In these cases, the alternation without stuttering allows k-induction to prove the unfeasibility of scenarios. If there exists a loop in the local structure of the automaton (i.e. a loop where all the transitions are labelled with a local event), due to the infinite nature of the state space we may have an infinite path in the system without loops, so that the simple-path condition never holds. In order to prove the unfeasibility of scenarios also in these cases, we combine k-induction with predicate abstraction as described in Section 6.3. We can associate to different segments of the MSC different abstractions of the local transition relation. This way, we can obtain a fine-grained abstraction that abstracts away the continuous components only where necessary.

7.4 Unfeasibility Explanation

In the case the CMSC $\langle m, \phi \rangle$ is unfeasible in the network $\mathcal{N}$ we provide the user with explanations that help to identify the reasons of the unfeasibility of the CMSC. In this section we first describe what kind of explanations we provide, their formalization and how we compute them. Then, we present three different case studies where we use the unfeasibility explanation to infer the cause of the unfeasibility of the scenario.

We identify the following three types of explanations:

1. an unfeasible prefix of the CMSC, which reduces the number of events and constraints to be inspected by a user to find a bug (either in the scenario or in the network);

2. why the paths of the network consistent with m cannot satisfy ϕ. The explanation is a formula that helps the user to understand the behaviors of $\mathcal{N}$ that are not consistent with ϕ;

3. why the paths of a component consistent with the corresponding instance of m are not consistent with the rest of m: this formula helps the user in detecting if a component is involved in the unfeasibility and, in that case, what synchronization constraints are not consistent with the other components in the network.

A *prefix* of a CMSC $\langle m, \phi \rangle$ is a CMSC $\langle \overline{m}, \overline{\phi} \rangle$ such that: $\overline{m} := \overline{\sigma}_1 || \ldots || \overline{\sigma}_n$ and m is consistent; for all $1 \leq i \leq n$, $|\overline{\sigma}_i| \leq |\sigma_i|$ and for all $1 \leq j \leq |\overline{\sigma}_i|$, $\overline{\sigma}_i[j] = \sigma_i[j]$; $\overline{\phi}$ contains only the conjunct of ϕ over the variables $V_{\overline{m}}$.

Given a CMSC $\langle m, \phi \rangle$ and a network $\mathcal{N}$ such that $\mathcal{N} \not\models \langle m, \phi \rangle$, the unfeasibility explanation of the *first type* is an unfeasible CMSC prefix $\langle \overline{m}, \overline{\phi} \rangle$ of $\langle m, \phi \rangle$.

Given a path $\pi_i \models \sigma_i$ and a tuple of bounds $\overline{k}_i := \langle k_i[0], \ldots, k_i[|\sigma_i|] \rangle$, where $k_i[j]$ is the length of a local segment, we say that π_i is *bounded by* $\overline{k}_i$ if for all $0 \leq j \leq |\sigma_i|$ the length of $lsg(\sigma_i[j])$ in π_i is $k_i[j]$. Given a path π of $S_{\text{LocTime}}(\mathcal{N})$ compatible with m and $\overline{k} := \langle \overline{k}_1, \ldots, \overline{k}_n \rangle$, where $\overline{k}_i$ is a tuple of bounds, we say that π is *bounded by* $\overline{k}$ iff for all $1 \leq i \leq n$, $prj(\pi, i)$ is bounded by $\overline{k}_i$.

Given a CMSC $\langle m, \phi \rangle$ unfeasible in $\mathcal{N}$ and bounds $\overline{k}$, an explanation of the *second type* is a formula ψ over V_m such that $\mathcal{N} \parallel m \models_{\overline{k}} \psi$ and $\psi \wedge \phi \models \bot$, where $\mathcal{N} \parallel m \models_{\overline{k}} \psi$ iff for all π of $S_{\text{LocTime}}(\mathcal{N})$ bounded by $\overline{k}$, $\bigcup_{1 \leq i \leq n, 0 \leq j \leq |\sigma_i|+1} pre_j(prj(\pi, i)) \models \psi$.

Given a CMSC $\langle m, \phi \rangle$ unfeasible in $\mathcal{N}$ and bounds $\overline{k}$, an explanation of the *third type* is a formula ψ_i over V_{σ_i} such that $S_i \parallel \sigma_{i_{|\phi_i}} \models_{\overline{k}} \psi_i$ and $\|_{j \neq i} S_j \parallel \sigma_{j_{|\phi_j}} \models_{\overline{k}} \neg\psi_i$. $S_i \parallel \sigma_{i_{|\phi_i}} \models_{\overline{k}} \psi_i$ iff for all paths π_i of S_i bounded by $\overline{k}_i$ $\bigcup_{0 \leq j \leq |\sigma_i|+1} pre_j(\pi_i) \models \psi_i$ and $\|_{i \neq j} S_j \parallel \sigma_{j_{|\phi_j}} \models_{\overline{k}} \neg\phi$ iff for $i \neq j$, for all π_j bounded by $\overline{k}_j$, $\bigcup_{i \neq j, 0 \leq v \leq |\sigma_j|+1} pre_v(\pi_j) \models \neg\psi_i$.

We extract the explanations by exploiting both unsatisfiable cores and interpolation. In particular, when we perform the inductive check incrementally on the CMSC $\langle m, \phi \rangle$ we run a satisfiability check after we have encoded all the synchronizations in the same partial order defined by $\leq_m$. If the encoding is unsatisfiable, by Theorem 12 we known that $\langle m, \phi \rangle$ is unfeasible. We extract the unsatisfiable core ξ of the encoding and different interpolants from the same proof of unsatisfiability.

In order to compute an unfeasible CMSC prefix $\langle \overline{m}, \overline{\phi} \rangle$, we use the unsatisfiable core ξ. ξ contains a subset of the encoding of the local paths, events, and constraints, since they are encoded in different conjuncts. Thus, ξ is fine-grained enough to obtain a precise subset X of the elements of the CMSC. For example, if the formula $Trans_{i|\sigma_i[j]}^{idx_i[j]} \wedge Inv_i^{idx_i[j]}$ is in ξ, it means that the encoding of the event $\sigma_i[j]$ is in X. The unfeasible CMSC prefix $\langle \overline{m}, \overline{\phi} \rangle$ is obtained taking all the elements of $\langle m, \phi \rangle$ that are in the relation $\leq_m$ with an element of X.

We compute the explanation of the second type ψ using interpolation. We partition the formulas in the unsatisfiable core ξ in two different formulas, A and B. A is the conjunction of all the formulas of ξ that encodes the parallel composition of the network $\mathcal{N}$ and the MSC m (i.e. $enc(m, \overline{k})$), while B contains the other formulas of ξ (i.e. $\phi[v_i^{idx_i[j]}/v_i[j]]$). ψ is an interpolant of A and B. Note that, if $\psi \models \bot$, we deduce that ϕ is not responsible of the unfeasibility and that the unrolling of the network is inconsistent by itself.

Finally, we compute the third explanation type ψ_i for the i-th component of $\mathcal{N}$. We partition the unsatisfiable core ξ in the formulas A and B. A is the conjunction of all the formulas of ξ that encode the unrolling of the i-th component along its instance σ_i and CMSC constraints (i.e. $enc(\sigma_i, \overline{k}_i)$ and ϕ_i), while B is the conjunctions of all the other formulas of ξ. If $\psi_i \models \top$, the component does not play a role in the unfeasibility. On the contrary, if $\psi_i \models \bot$, the component does not have a path compatible with its instance.

Note that, when the abstraction is used to prove the unfeasibility of the scenario, the three types of explanations are still valid.

Distributed Controller [HH94]. This benchmark models the interactions of two sensors (sensor$_1$ and sensor$_2$) with a controller of a robot. The two sensors interact with a scheduler to access a shared processor. The time needed for computation by the two sensors is bounded but it is non-deterministic, and is tracked in the scheduler with two stopwatches (x_1 and x_2). Also, the controller sets a time-out (variable $z = 0$) after the receipt of the first message. If the time-out expires ($z = 10$) the controller discards all the received data.

The MSC shown in Figure 7.3 models the interaction where sensor$_1$ requests the processor; the scheduler grants it for a total duration of x_2 time; sensor$_2$, which has a higher priority, requests and receives grant to the processor; when sensor$_2$ finishes its computation (event *read*$_2$), sensor$_1$ accesses the processor (event *read*$_1$); in parallel, sensor$_2$ sends its data to the controller; finally, the sensor$_1$ and the controller synchronize on *send*$_1$ and *ack*$_1$. The time spent to process the data of sensor$_1$ is given by the stopwatch x_1. In Figure 7.3 x_1 is the sum of the intervals x'_1 and x''_1. Moreover, we add two additional conditions on the duration of x_1 and x_2 in the scheduler ($x_2 = 1.5$ and $x_1 = 1.1$), and we fix the maximum time spent by the controller before receiving the data from sensor$_1$ ($z < 1$). The MSC augmented with these constraints is unfeasible.

The scenario is proved unfeasible. In Figure 7.3 we outline in gray the elements of the scenario, events and constraints, which contribute to the unfeasibility. In particular, the unsatisfiable core contains all the events of the CMSC apart from *Ack*$_1$ and *Ack*$_2$. Thus, the unfeasible CMSC prefix does not include the last event *Ack*$_1$.

We exploit the interpolation techniques to get the constraints $z >= x_1$.All the interpolants are computed by the SMT solver, then we man-

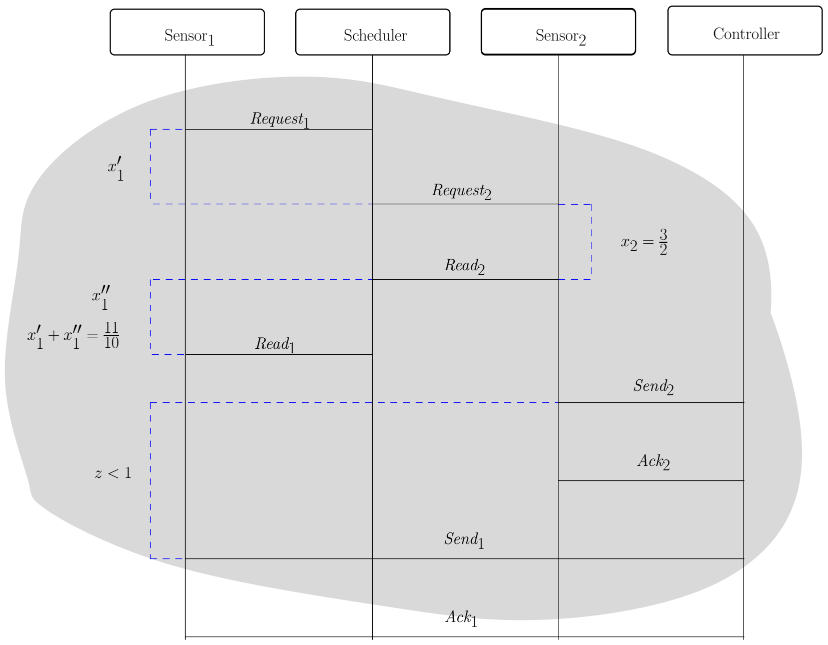


Figure 7.3: The MSC for the distributed controller.

ually simplify them removing redundant constraints. In fact, z counts the time elapsed in the controller between the send$_1$ event and the send$_2$ event. This means that the controller cannot receive the send$_1$ message before x_1 seconds, which is the time spent to process data from sensor$_1$. If we fix $z >= 1.1$ then the scenario is feasible. We find a similar result if we look at the interpolant obtained partitioning the encoding in the constraints from sensor$_1$ (the A formula) and the rest of the network and the scenario (the B formula). We denote with $time^{event}_{\text{component}}$ the time variable of *component* when performing *event*. The interpolant is $6 <= time^{request_1}_{\text{sensor}_1} - time^{read_1}_{\text{sensor}_1} + time^{send_1}_{\text{sensor}_1}$. Since $time^{request_1}_{\text{sensor}_1}$ is 6, from the initial condition and invariants of sensor$_1$, we can infer that the scenario and the other processes in the network do not allow $time^{read_1}_{\text{sensor}_1} <= time^{send_1}_{\text{sensor}_1}$, which is a necessary condition for sensor$_1$.

Audio Control Protocol [HH94]. The benchmark models a protocol that transmits an arbitrary-length bit sequence from a sender to a receiver based on the timing-based Manchester encoding. The protocol relies on division of the elapsed time in slots. Every slot corresponds to a bit. The sender transmits a signal *up* in the slots corresponding to bits with value 1 (thus, a slot without signals correspond to bit 0). The protocol is robust to bounded errors in the timers used by the sender and receiver.

The considered scenarios consist of unfeasible timed sequences of *up*. For example, the sequence $\langle up, 4\rangle$, $\langle up, 8\rangle$, $\langle up, 12\rangle, \langle up, 16\rangle, \langle up, 19\rangle, \langle up, 23\rangle$ does not respect the protocol, since the 4-th and 5-th events must be separated by 3 seconds.

Once scenario-based induction proves that the scenario is unfeasible, the unsatisfiable core contains the encoding of the 4-th and 5-th event, thus the unfeasible CMSC prefix is the CMSC formed by the first 5 events. Interpolation "explains" that the inconsistency arises because the sender requires the 5-th event to happen after at least 3.8 seconds; it also shows that the receiver does not play any role in the inconsistency.

Electronic Height Control System [MS00] . This industrial case study models a system that controls the height of a car's chassis. A timer tells the controller when to read the height from a filter, while disturbances which changes the height of the vehicle are modeled by the environment. The MSC describes a scenario where the height of the chassis falls outside the allowed thresholds, first below and then above the permitted height intervals.

The scenario is not feasible due to the timing constraints imposed by the timer on each event and to the dynamics of the environment which requires an incompatible time to pass from the initial level of the chassis to a value read outside the allowed threshold. More precisely, the timer forces every event to happen every second, while the filter chassis level f read by the sensors evolves according to the differential equation $\dot{f} = \frac{h-f}{T}$, where h represents the current level. This is approximated by the linear-phase portrait partitioning which linearizes the differential equation into flow conditions of the form $\dot{f} \in [a, b]$.

In order to prove the scenario unfeasible, abstraction is required. In fact, in the concrete state space of the environment, the portrait partitioning creates discrete loops that correspond to infinite simple paths, and thus concrete K-induction can not converge. K-induction combined with abstraction proves that the controller and the timer do not have a simple path longer than 1 alternating timed and discrete transitions (since there is no local transition).

For the environment we used a set of predicates in the form $t \in [i, i+1]$, $h \in [at, bt]$ and $f \in [at, bt]$ where i is an integer while a and b are the constants used in the partitioning. We localize the abstraction by using $t \in [i, i+1]$ only in the i-th event and considering the partition consistent with the initial, values. We added a total of 16 predicates to the first two local segments of the environment. This way, the unfeasibility can be proved unrolling the environment for 25 steps in the first segment and 20 steps in the second segment. The details of the abstraction are available at `http://es.fbk.eu/people/mover/tests/FMSD11/`.

We extract an unfeasible CMSC formed by the first 2 events. Non-trivial explanations of the third type are associated with the timer, where the 2-nd event must happen in less then 3 seconds, and with the environment, where the same event to happen not before 3.3 seconds.

7.5 Related work

MSCs [IT96] are a basic building block to describe the interactions among components.
Several works, such as High-Level Message Sequence Charts [MR97] and Live Sequence Charts (LSC) [DH01], extend the language of the MSCs increasing their expressive power. We consider a basic version of MSCs which describes a single (partial-order) composition of sequences of events, augmented with additional constraints [ABG07, BAL97, CM06]. We consider a trace-based semantics for the MSC, where the MSC predicates range over the observable events of a system [LL92, LL95]. While several works use MSCs to describe the entire system [AY99, PBL09], we instead use the MSC as a specification language.

A common approach to deal with the verification of MSC specifications consists in translating the scenario into automata or temporal logic formulas. In [CM06] the authors consider the feasibility problem for MSCs with timed constraints and a timed message-passing automaton. Their solution consists of a translation of the timed MSC into an automaton, reducing the problem to a reachability analysis for timed systems. They support also "weak" embeddings, where the MSC specification can be partial. *Live Sequence Charts (LSCs)* are translated into timed automata in the UPPAAL model checker [LBD$^+$10], while in [KW01] the authors propose a translation from charts with timing constraints and synchronous events to Timed Büchi Automata. These works deal with expressive specification languages but they do not exploit the structure of the scenario. Moreover, in case of unfeasibility, these techniques do not provide explanations that narrow the events of the scenario or that give meaningful information about a specific component.

The approach which translates the MSC into an automaton reduces the feasibility problem of the MSC to a reachability problem. Thus, the works on Bounded Model Checking (BMC) for hybrid systems [ABCS05, FH05, FH07, BCL$^+$10a, ÁBKS05] can be used to solve the feasibility problem. The use of the step semantics to optimize the BMC encodings for asynchronous systems was investigated in [DJH12, HN03]. However, BMC is unable to prove the unfeasibility of the MSC. When we encode the MSC into an automaton the unfeasibility problem can be solved using unbounded model checking techniques, such as k-induction [SSS00]. K-induction is complete for finite state systems, but it was applied also to infinite state systems in [dMRS03, Ton09, Pik05]. In [dMRS03] the authors use k-induction to verify timed and hybrid automata and they generalize the simple path condition to simulation relations. K-induction is combined with predicate abstraction in [Ton09]. These works are not tailored to the problem of deciding the unfeasibility of a scenario and do not provide explanations in the case of unsatisfiability.

Unsat cores and interpolation are often used to explain and generalize the source of unsatisfiability. Unsat cores are typically subsets of the conjuncts forming the unsatisfiable formula. However, other forms are possible, especially in the context of temporal unsatisfiability [Sch12, Sch09b]. Interpolation for temporal properties is proposed in [SV07] as a theoretical framework for analyzing vacuity for discrete systems; the practical implications are not addressed in depth. In [Sch09b], it is suggested that k-induction can be used to find a k for which the BMC encoding of a temporal formula yields its unsatisfiability and that the unsat core contains the relevant parts of the formula that cause the unsatisfiability. However, mapping the BMC unsat core back to the original problem is not always easy. We achieve this by exploiting the scenario-based encoding that respects the structure of the scenario.

Chapter 8

Parameter synthesis

Note. The material presented in this chapter has already been presented in [CGMT13].

Parametric systems arise in many application domains from real-time systems to software to cyber-physical systems. Parameters allow to model several features of the designed system. For example, they can model unknown physical constants of the environment or other constants that the designer must define of the system (e.g. timing constraint of real-time systems). Moreover, in the context of hybrid systems, it is important to guarantee the robustness with respect to the choice of parameters, that can be in a range of values instead of being a fixed constant. Thus, the use of parameters in the early phases of the development gives the possibility to explore different design choices. In fact, note that a parametric system represents a set of (non-parametric) systems, one for each valuation of the parameters.

A key challenge for the design of parametric systems is the estimation of the parameter valuations that guarantee the correct behavior of the system (e.g. that the model satisfies an invariant property). The manual estimation of the parameters values is a time-consuming task, that does not allow the designer to either find the optimal values of the parameters (this is useful to reduce costs or increase the performance of the model) or to guarantee that the system behaves correctly for a specific range of parameters values. Thus, there is the need of automatic techniques that solve the parameter synthesis problem.

In this chapter, we present a parameter synthesis algorithm that is targeted to infinite-state transition systems and invariant properties. While the scope of the technique is general, it can be applied off-the-shelf to the symbolic encoding of of an hybrid system.

The algorithm is based on a general SMT-based approach for parameter synthesis that works by complement, building the set of "bad" parameter valuations (i.e. the set of parameters valuations that may violates the invariant property) It relies on the enumeration of counterexamples violating the properties, extracting from the counterexample a region of bad parameter valuations by quantification of the state variables.

The novel synthesis algorithm is based on IC3, one of the major recent breakthroughs in SAT-based model checking, and lately applied to the SMT case. The key idea of the synthesis algorithm is to exploit the features of IC3. First, IC3 may find a set of counterexamples consisting of a sequence of set of states $s_o, \dots, s_k$, where each state in s_i is guaranteed to reach some of the bad states in s_k in k steps; this is exploited in the expensive quantification of the state-variables, that can be performed on shortest, and thus more amenable, counterexamples. Second, the internal structures of IC3 allows our extension to be integrated in a fully incremental fashion, never restarting the search from scratch to find a new counterexample.

Various approaches already solve the parameter synthesis problem for several kind of systems, like infinite-state transition systems [BCGR12] and timed and hybrid automata [HH94, Wan05, FJK08, CPR08, AK12, BLR05]. The advantages of the new algorithm are that it synthesizes an optimal region of parameters (unlike [FJK08, AK12]), it is incremental and applies quantifier elimination only to small formulas (unlike [FJK08, CPR08]), and it avoids computing the whole set of the reachable states (unlike [HH94, Wan05]).

In the following, we formally introduce the parameter synthesis problem in Section 8.1 and the general approach based on reachability and quantifier elimination in Section 8.2. Then, we show our new approach based on IC3 in Section 8.3 and we conclude the Chapter describing some related work 8.4.

8.1 Problem definition

In this chapter we use the transition system definition without input variables and the invariant formula.

In parametric systems, besides the standard constants, the formulas can include also *parameters*, which are rigid symbols with "unknown" values. Let U be the set of parameters. A *parameter valuation* is as assignment to the parameters. Given a formula ϕ and a parameter valuation γ, we denote with $\gamma(\phi)$ the formula obtained from ϕ by replacing each parameter in U with the assignment given by γ.

Definition 25 (Parametric transition system) *A* parametric transition system *S is a tuple $S = \langle U, V, Init, Trans\rangle$ where U is the set of parameters, V is the set of variables, $Init(U, V)$ is the initial formula, and $Trans(U, V, V')$ is the transition formula. Each parameter valuation γ induces a transition system $S_\gamma = \langle V, \gamma(Init), \gamma(Trans)\rangle$.*

Given a parametric transition system $S = \langle U, V, Init, Trans\rangle$ and a formula $P(U, V)$, we say that a parameter valuation γ is *feasible* iff $S_\gamma \models \gamma(P)$.

Definition 26 (Parameter synthesis problem) *The* parameter synthesis problem *is the problem of finding the set $\rho(U)$ of all the feasible parameter valuations (i.e., for every $\gamma \in \rho$, $S_\gamma \models \gamma(P)$).*

Thus, we solve the *optimal* parameter synthesis problem, where we find the set of feasible parameter valuations $\rho(U)$ that contains all the feasible parameter valuations.

8.2 Solving the synthesis problem with reachability

A naive approach to synthetize the set of parameters $\rho(U)$ is to incrementally find the complement set $\beta(U)$ (thus, $\rho = \neg\beta$) of unfeasible parameter valuations rephrasing the problem as a reachability problem for a transition system S_ρ and iteratively removing the counterexamples to $S_\rho \models P$.

More specifically, given the parametric transition system $S = \langle U, V, Init, Trans\rangle$, the algorithm keeps an over-approximation $\rho(U)$ (initially true) of the safe region. The encoding of S is the transition system $S_\rho = \langle V \cup P, Init_\rho, Trans_\rho\rangle$ where $Trans_\rho = Trans \wedge \bigwedge_{p \in U} p' = p$ forces parameters to not change their value in the evolution of the system and $Init_\rho = Init \wedge \rho$ restricts the parameter valuations to the over-approximation.

At every iteration, a new parameter valuation is removed from ρ. The algorithm terminates if it proves that $S_\rho \models P$, and ρ is the solution to the synthesis problem.

This simple approach does not work in the context of infinite-state transition systems, where in general the possible number of counterexamples and the values of the parameters are infinite. For this reason, we need an algorithm that removes a set of parameters, instead of a single point.

8.3 Description of the synthesis algorithm with IC3

We embed a reasoning similar to the naive algorithm in IC3, exploiting the generalization of counterexamples and the incremental behavior. The generalization avoids the explicit enumeration of the counterexamples, while the incrementality allows us to *completely* reuse all the clauses learned by IC3 across different safety checks.

Therefore, IC3 is used to prove that $S_\rho \models P$. If it is successful (recall that in the SMT extension, the problem is undecidable), we can conclude that ρ is a set of feasible parameters and, in particular, is optimal. Instead, if there exists a set of parameters such that $S \not\models P$, IC3 will might find a counterexample to P. The counterexample is found in the blocking phase as a sequence $\pi := (s_0, 0); \ldots; (s_n, n)$, where $s_0 \models Init$, $s_n \models \neg P$ and for $0 < i < n{-}1$, $s_i {\wedge} Trans \models s_{i+1}$. Possibly, π does not represent a single path of the system that reaches a violation, but a set of paths that reach $\neg P$. This is an intrinsic feature of IC3, which generalizes the counterexamples to induction found in the blocking phase, trying to block set of states rater than a single state. The state s_o represents a set of states that will eventually reach $\neg P$. Thus, we compute from s_0 a set of bad parameters $\beta_{s_o}(U)$ that will eventually reach s_n: $\beta_{s_o}(U) := \exists V. s_o(U, V)$. We rely on a quantifier elimination procedure to obtain a quantifier-free formula for β_{s_o}.

The algorithm refines its conjecture about the unfeasible parameters of the system. Let $\beta' := \beta \vee \beta_{s_o}$ and $\rho' := \rho \wedge \neg\beta_{s_o}$ be the new approximations of unfeasible and feasible regions of parameters. We have to prove that $S_{\rho'} \models P$. We perform the verification incrementally, reusing all the frames of IC3. Since $\rho' := \rho \wedge \neg\beta_{s_o}$, we have that $S_{\rho'} = \langle V \cup P, Init_\rho \wedge \neg\beta_{s_o}, Trans_\rho \rangle$.[1] Thus, we incrementally encode $S_{\rho'}$ strengthening the initial condition and the transition relation used in the algorithm,

[1]We also add $\neg\beta_{s_o}$ also to $Trans_\rho$, since it is an inductive invariant of $S_{\rho'}$.

and also strengthening the first frame kept by the IC3 algorithm (i.e. $F_0 := F_0 \wedge \neg(\beta_{s_o}))$. The strengthening of F_0 removes the state s_o from $Init$ (possibly blocking also other bad states).

Since S_ρ is an overapproximation of $S_{\rho'}$, the invariant kept by IC3 (i.e. $F_0 = Init$, $F_i \models F_{i+1}$, $F_i \models P$ and $F_i(V) \wedge Trans(V, V') \models F_{i+1}(V')$) also holds for the new problem $S_{\rho'} \models P$.

From this point, we rely on the usual behavior of IC3, which tries to block $(s_1, 1)$ with the strengthened frame F_0. The algorithm terminates if either P is proved or the F_0 becomes unsatisfiable, showing that ρ is empty.

We show the parameter synthesis algorithm ParamIC3 in the Figure 8.1, highlighting in red the modifications to the original IC3.

Theorem 13 *Given a parametric transition system $S = \langle U, V, Init, Trans\rangle$ and a formula $P(V)$, $\rho(U) :=$ ParamIC3$(U, Init, Trans, P)$ is the optimal set of feasible parameter valuations.*

Proof. The proof is by induction on the number of iterations of ParamIC3. The proof shows that if the algorithm terminates, it returns the optimal region of parameters.

Consider the first iteration of the algorithm.

Suppose we prove $S_\rho \models P$. Then, we know that there are no bad parameter valuations such that S_ρ is not safe. Also, $\rho = \top$ is the optimal parameter region.

Instead, suppose we find $\pi = s_0; \ldots; s_n$, a counterexample to $S_\rho \models P$. The counterexample π found by IC3 is such that for each state u in s_0 there exists a path $u_0; \ldots; u_n$, where $u_n \models \neg P$. We compute the set of bad parameters $\beta_{s_0}(U) := \exists V.s_0(U, V)$. From π, we know that for all parameter valuations $\gamma \in \beta_{s_o}$, there exists a state u_0 in s_0 (i.e. $u_0 \models s_0$) and a path $u_0; \ldots; u_n$ such that $\gamma \models u_0$ and $u_n \models \neg P$. Thus, $S_{\rho_\gamma} \not\models P$ for all $\gamma \in \beta s_0$.

Intuitively, it means that ParamIC3 only removes valuations of unfeasible parameters.

The new under-approximation of bad parameters is $\beta' := \beta(s_0)$, while the over-approximation of the good parameters is $\rho' := \rho \wedge \neg\beta(s_o)$. We consider the transition system $S_{\rho'}$ and the new problem $S_{\rho'} \models P$. Then, the algorithm set the following variables (here, we use the primed notation to refer to the value of the variables after the assignment): $Init_{\rho'} := Init \wedge \neg\beta(s_o)$, $Trans_{\rho'} := Trans \wedge \neg\beta(s_0)$, $F'_0 := Init \wedge \neg\beta(s_o)$. The invariants on the IC3 traces (the one presented in the Subsection 2.3.1) holds for checking the problem $S_{\rho'} \models P$:

- $F'_0 = Init_{\rho'}$: it holds, since $F'_0 := Init \wedge \neg\beta(s_o) = Init_{\rho'}$;
- $F'_i \models F_{i+1}$: it holds, since $F'_0 \models F_1$;
- $F'_0 \models F_1$ and for $1 \leq i \leq n$, $F_i(V) \wedge Trans_{\rho'}(V, V') \models F_{i+1}(V')$.
  Consider $F'_0(V) \wedge Trans_{\rho'}(V, V') \models F_{i+1}(V')$.
  We have to prove that $F'_0(V) \wedge Trans_{\rho'}(V, V') \wedge \neg F_{i+1}(V') \models \perp$. By hypothesis, we know that $F_0(V) \wedge Trans_{\rho}(V, V') \wedge \neg F_{i+1}(V') \models \perp$. Suppose there exists a model μ such that $\mu \models F'_0(V) \wedge Trans_{\rho'}(V, V') \wedge \neg F_{i+1}(V')$. Recall that $F'_0(V) \wedge Trans_{\rho'}(V, V') \wedge \neg F_{i+1}(V')$ is $F_0(V) \wedge (\neg\beta_{s_0}) \wedge Trans_{\rho}(V, V') \wedge (\neg\beta_{s_0}) \wedge \neg F_{i+1}(V')$. Thus, $\mu \models F_0(V) \wedge Trans_{\rho}(V, V') \wedge \neg F_{i+1}(V')$, contraddicting the hypothesis. By a similar reasoning, we can prove that $F_i(V) \wedge Trans_{\rho'}(V, V') \models F_{i+1}(V')$ holds, for $1 \leq i \leq n$.
- for all $i < k$, $F_i \models P$: it holds, since $F'_0 \models P$.

Now, suppose we are at the $n - th$ iteration, where ρ_n and β_n are the approximations of good and bad parameters found so far.

- Suppose we prove $S_{\rho_n} \models P$. Thus, we proved that ρ_n is a region of good parameters.

Note that, by induction every bad region found by ParamIC3 $\beta_1 \wedge \dots \beta_n$ contains only unfeasible parameter valuations and $\rho_n := \neg(\beta_1 \vee \dots \vee \beta_n)$. Thus, ρ_n is optimal.

- Instead, suppose we still find a counterexample $\pi = s_0; \dots; s_n$. We compute $\beta(s_0)$, which is a set of bad parameter valuations for S_{ρ_n}. We can prove that $\beta(s_0)$ only contains valuations for "bad" parameters applying the same reasoning done in the first iteration.

  We have that $S_{\rho_{n+1}} = S_{\rho_n} \wedge (\neg\beta(s_0))$. With a reasoning similar to the one that we did in the fist iteration case, we can prove that the invariant on the frames of IC3 holds for the model checking problem $S_{\rho_{n+1}} \models P$.

$\diamond$

8.3.1 Optimization

We presented a version of the algorithm which computes a region of bad states β_{s_o} only from the initial states of $\pi := s_0; \dots; s_n$. However, this is only one of the possible choices, since more general regions of bad parameters can be found considering each s_i in π. In fact, β_{s_o} is one of the extreme cases, while the other one is $\beta_n(U) := \exists V.(BMC(n))$, which encodes the set of all the parameters that may reach $\neg P$ in n steps, where $BMC(n)$ denotes $Init^0 \wedge \bigwedge_{i=0}^{n-1} Trans^i \wedge \neg P^n$. However, the cost of eliminating the quantifiers grows as well, and it might in fact become impractical. In principle, one may consider the intermediate cases β_{s_i} (that is, the reachability of one of the intermediate states s_i in π) to trade the generality of the result with the cost of the quantifier elimination. Furthermore, we notice that for soundness we do not need the precise set β_{s_i}, but we can consider its under-approximations, since this still guarantees to remove only bad parameters valuations. As an advantage, in this case the quantifier elimination problems are easier to solve and are more general than β_{s_o}. In practice, we use an heuristic which tries to combine the precise and the under-approximated approach, enabling us to find a trade-off between generality and the cost of quantifier elimination. The heuristic that we use is described in the experimental evaluation (Section 10.5).

8.4 Related work

The IC3 [Bra11] algorithm was first proposed to prove safety property for finite-state transition systems. Several approaches adapted the original algorithm to deal with infinite-state systems [CG12, HB12, KJN12b].

The works presented in [CG12, HB12] may be used as backends to synthesize the parameters via reachability. However, they need to perform a quantifier elimination step on an entire path and they will not be able to exploit the information discovered by IC3 while finding violations to the property. Instead, [KJN12b] cannot be used as a reachability backends, since it is restricted to timed automata without parameters.

The parameter synthesis for infinite-state transition systems can be solved combining a reachability algorithm with a quantifier elimination procedure, as proposed by [CPR08]. While the approach was proposed in the context of parametric timed automata, it may applied to infinite-state systems. Our approach follows the same general idea, which is to iteratively find the unfeasible regions of parameters. However, a key difference is in the computation of a set of bad region with the quantifier elimination procedure, since we apply the quantifier elimination to a set of states and not to a set of traces. The approach proposed in [BCGR12] deals with infinite-state systems with an unbounded number of processes and several kind of properties, like mutual exclusion and deadlock detection. While this setting is more general than ours, it does not synthesize the entire region of parameters, but it instantiates the values of the parameters for a given template.

Other works [HH94, Wan05, FJK08, CPR08, AFKS12, AK12, GSV13] synthesize parameters for real-time and hybrid systems.
Tools like HyTech [HH94], TReX [ABS01] or Red [Wan05] synthesize the parameters computing the reachable states of the system. All these techniques are precise, but they are forced to compute the entire set of reachable states, which may be unfeasible in several cases.

The technique proposed by Frehse [FJK08] handles linear hybrid automata, using an approach similar to [CPR08]. The approach is not precise and underapproximates the region of feasible parameters. Instead, we find the precise region of parameters.

The tools IMITATOR [AFKS12] and HYMITATOR [AK12], which respectively handle timed and hybrid systems, solve a different but related problem to parameter synthesis. Given a parameter valuation, they compute all the parameter values that induce the same set of discrete traces as the given parameter valuation. This approach requires an initial assignments for the parameters and in general it will not find the maximum region of feasible parameters.

Finally, the work [GSV13] synthetize specific parameters (sensing and actuation intervals) for *lazy linear hybrid automata* (LLHA) [JBS07]. The approach uses BMC as primitive to find the two intervals, using a search based on bisection. The approach is specifically designed for the control strategy problem for LLHA and it relies on the fact that LLHA are finite, due to discretization. The assumption is exploited to fix a bound of the BMC unrolling and to encode the problem to theories different from rational numbers, like the *Theory of Bit-Vector*. Our technique is more general, since it can synthetize non-convex regions. However, our current implementation is specialized to handle problems in the *Linear Arithmetic* *over Rationals*, since our implementation relies on a quantifier elimination procedure for that theory. In principle it could be applied off-the shelf to synthetize control strategies for LLHA through bit-blasting.

We stress that our approach is not specific to timed and hybrid automata, but it may be applied to every infinite-state transition systems expressed using *Linear Rational Arithmetic*.

```
bool ParamIC3 ($U$, $Init$, $Trans$, $P$):
$\beta(U) = \bot$  # underapproximation of the unfeasible parameters
trace = [$Init$]  # first elem of trace is init formula
trace.push()  # add a new frame to the trace
while True:
    # blocking phase
    while there exists a cube $c$ s.t. trace.last() $\wedge$ $Trans$ $\wedge$ $c$ is satisfiable
            and $c \models \neg P$:
        recursively block the pair $(c, \text{trace.size}() - 1)$
        if a pair $(p, 0)$ is generated:
            $\beta_p = \exists V.p(U, V)$
            $\beta := \beta \vee \beta_p$.
            $Init := Init \wedge \neg\beta_p$ and $Trans := Trans \wedge \neg\beta_p$ .
            add $\neg\beta_p$ to trace[0].
            remove $(p, 0)$ from the set of states to be blocked.
            if $Init \models \bot$  # the initial states are empty
                return $\bot$

    # propagation phase
    trace.push()
    for $i = 1$ to trace.size() $- 1$:
        for each clause $c \in$ trace[i]:
            if trace[i] $\wedge$ $c$ $\wedge$ $Trans$ $\wedge$ $\neg c'$ is unsatisfiable:
                add $c$ to trace[i+1]
        if trace[i] == trace[i+1]:
            return $\neg\beta$  # P proved, return good params region
```

Figure 8.1: High-level description of ParamIC3. The bold and red text shows the code which differ from the IC3 algorithm used for verification.

Part IV

Tools and Experimental Results

Chapter 9

HyCOMP

Note. Part of the material presented in this chapter has already been presented in [CMT11b].

HyComp is a model checker for asynchronous hybrid systems. HyComp analyzes hybrid systems specified in the HyDI input language. HyDI allow an user to model asynchronous hybrid systems with different dynamics (*Linear Hybrid Automata*, *Linear Hybrid Systems*, *Polynomial Hybrid Systems*) and to analyze them with different verification algorithms. HyComp allows to verify invariant properties, Linear Time Temporal Logic (LTL) properties, and scenario specifications. Moreover, the tool allow to perform other kinds of analysis, such as parameter synthesis.

HyComp is based on the nuXmv [nux] model checker, which in turn it is based on the NuSMV model checker [CCG+02], and the MathSAT SMT solver [CGSS13]. The tool was used in several projects. In the MISSA (More Integrated Systems Safety Assessment) project [MIS] HyComp was used to support the translation of industrial designs written in the Altarica language [BCL+11], while in the IRONCAP (Innovative Rover Operations Concepts Autonomous Planner) project the tool was used for the modeling and the validation of planning problems.

9.1 Tool features

HyComp provides several functionalities.

Infinite-state transition systems encoding of Hybrid Automata Network The first functionality of HyComp is to encode a network of hybrid automata in an infinite-state transition system. HyComp implements the encoding techniques presented in Part II.

HyComp encodes linear hybrid automata as shown in the Example 2. Related to the dynamic of the system, HyComp constructs automatically the quantifier-free encoding of Chapter 3. Currently, the encoding is implemented for the class of *Polynomial Hybrid Systems* (See Section 3.3) and for the class of *Linear Hybrid Systems* in the case the matrix of the system is nilpotent (See Section 3.4.1). Then, HyComp implements the time-aware relational abstraction technique of Chapter 5.

HyComp handles a network of hybrid automata. With respect to network, HyComp may generate either the *global-time* encoding or the *local-time* encoding presented in Chapter 4.

The tool has several options, that allow to tune the resulting encoding. For example, the user can choose to substitute the state variable that keeps track of the elapsed time, and encode only the amount of time elapsed with a single (input) variable δ (See Remark 2). Another option forces that a path in the resulting encoding can never have two consecutive continuous transitions. This optimization may be useful to increase the performance of the model checking tasks (See [ÁBKS05]) or to help inductive verification algorithms such as k-induction. Other options refer to the encoding of the synchronization constraints. The user can choose to use the "step semantic" variant of the synchronization's encoding.

The encoding is performed compositionally for each automaton in the network. This allow us to produce intermediate results in the encoding (e.g. a set of discrete transition systems that represent the encoding of the hybrid automata in the network). This feature is important to develop the algorithms based on the structure of the network, such as "Shallow Synchronization" and the scenario-based verification algorithms.

The resulting transition systems are either used internally by the HyComp algorithms or are used by the nuXmv algorithms, since the transition systems can be printed using the concrete syntax of nuXmv. This allow us to exploit all the capabilities of the nuXmv model checker.

HyDI currently allows limited verification capabilities for non-linear hybrid automata. Since the encoding relies on non-linear real arithmetic, the tool needs to use an SMT solver that handle this kind of theory. At the moment we interfaced only a version of Bounded Model Checking and induction (1-step induction) with such kind of SMT solvers. In practice, HyComp calls the Z3[1] and the iSAT[2] SMT solvers.

At the moment of this writing HyComp is not officially released. The tool will be released publicly and will be available at `https://es.fbk.eu/tools/hycomp/`.

Invariant model checking The main task of HyComp is to model check invariant properties.

One flow of verification uses the algorithms internally implemented in HyComp. These algorithms exploit the features of the input model, such as the continuous evolution or the structure of the network. HyComp implements the Bounded Model Checking algorithm based on "Shallow Synchronization" (Section 6.2), and variants for both BMC and k-induction where the discrete and the continuous transitions are alternated in the

[1]http://research.microsoft.com/en-us/um/redmond/projects/z3/
[2]http://isat.gforge.avacs.org/

BMC algorithm (i.e. the algorithm uses the transition relation restricted to the discrete transitions in the odd steps and the transition relation restricted to the continuous transition in the even steps of the encoding).

The alternative flow of verification exploits the invariant algorithms for infinite state systems implemented in the nuXmv model checker. To both falsify and prove an invariant property, nuXmv implements several SMT-based algorithms: IC3 [CG12], a version of IC3 that integrates implicit predicate abstraction [CGMT14a], interpolation-based model checking [McM03], k-induction and k-induction with implicit predicate abstraction (See Section 6.3). To falsify an invariant property, nuXmv implements Bounded Model Checking.

LTL model checking HyComp can check LTL properties (without the next operator) interpreted over the discrete sequences of states.

First, HyComp can use the LTL model checking features of nuXmv. nuXmv implements Bounded Model Checking for LTL properties[3]. Then, nuXmv implements a version of the k-liveness algorithm [CS12b].

Since k-liveness may be ineffective for hybrid systems, HyComp extends k-liveness to prove LTL properties for hybrid systems [CGMT14b].

Scenario-based verification HyComp implements all the scenario verification presented in Chapter 7. The Constrained Message Sequence Chart (CMSC) can be written in a specific input language. Then, HyComp allows to check both the feasibility and the unfeasibility of the CMSC, using the specialized algorithms or constructing a monitor.

Parameter Synthesis nuXmv implements the parameter synthesis algorithm described in Chapter 8. Thus, HyComp can be used to perform

[3]Note that, for infinite-state transition systems, this algorithm cannot find a counterexample that is not lasso-shaped.

parameter synthesis for *Linear Hybrid Automata*.

9.2 Tool architecture

HyComp is written in C and is about 50000 lines of code. From a high level point of view, HyComp uses both the nuXmv and the NuSMV model checkers as libraries.

NuSMV provides the basic functionalities to represent formulas, symbols with their types and state machines. Then, NuSMV implements several verification algorithms for finite-state transition systems based on SAT solvers and Binary Decision Diagrams (BDDs). Also, NuSMV wraps the CUDD BDD package (`http://vlsi.colorado.edu/~fabio`) functionalities. HyComp currently uses these functionalities to analyze abstractions in the case of scenario-based verification.

nuXmv provides several verification algorithms both for finite-state and infinite-state systems (e.g. IC3) and an interface to the MathSAT SMT solver.

Internally, HyComp is divided in several sub-packages.

- *Parser*: the package implements the parsing of HyDI models.
- *Network*: the package provides the data structures used to represent hybrid automata networks and also networks of symbolic transition systems.
- *Encoding*: the package implements the encoding of hybrid automata as symbolic transition systems and the local-time and global-time encodings.
- *Scenario*: implements the scenario verification algorithms.

- *BMC*: implements the Bounded Model Checking algorithm of HyComp.
- *LTL*: implements the LTL verification algorithm based on k-liveness.
- *CMD*: implements several utility commands that wraps the nuXmv verification algorithms. This allow an user to call the algorithm in HyComp, without using nuXmv or NuSMV.

9.3 The HyDI language

The HyDI language can be used to model a network of symbolic hybrid automata. HyDI extends the language of the nuXmv model checker [nux], that in turns extend the language of NuSMV [CCG$^+$02]. nuXmv extends the NuSMV language with infinite types, like *integer* and *real*. The nuXmv language can represent synchronous, symbolic infinite state transition systems. Thus, it cannot represent hybrid systems and asynchronous systems. HyDI extends the nuXmv language in two main directions:

1. *Hybrid Automata:* HyDI is interpreted with a continuous time semantic, introducing continuous type variables and constructs that define the continuous evolution with differential equations.
2. *Asynchronous systems:* HyDI can represent a network of symbolic hybrid automata, which move asynchronously and synchronize via message passing (or via shared variables).

In the following section we give an informal overview of the language. A deeper insight about the concrete syntax may be found in the HyDI language tutorial [CMT13a]. Then, we give a formal characterization of the language, giving its abstract syntax and semantics.

9.3.1 Overview of the language

A HyDI model is given by a set of *modules*, a set of *processes*, and a set of *synchronization* constraints. Figure 9.1 shows a small example of the HyDI specification of two gates. A gate is open, closed, it is opening or closing. Both gates must open and close together.

Modules

HyDI modules (e.g., the `Gate` module) extend SMV modules in order to specify explicitly the events used for the synchronization and timing aspects, such as continuous variables and flow conditions. The SMV language has been widely used to specify complex finite-state systems. The system description is typically decomposed into modules. Essentially, a module is a set of declarations and constraints on the declared variables. Modules can be instantiated several times and nested to form a complex synchronous hierarchy.

In particular, modules may contain `VAR` sections with the declaration of state variables (the states of the system consist of assignments to these variables); `IVAR` sections with the declaration of input variables; `INIT` constraints which must be satisfied by the valid initial states; `INVAR` constraints which must be satisfied by any valid state; and `TRANS` constraints which must be satisfied by any valid transition. For example, the state $s_1 = \langle location = opened, timer = 10\rangle$ is a valid state but not initial; it can go to the state $s_2 = \langle state = closing, timer = 0\rangle$ but not to the state $s_3 = \langle state = closing, timer = 10\rangle$ (because the transition would violate the constraint $next(timer) = 0$).

HyDI modules inherit all the constructs of SMV modules and add three main new features:

- *events*, a list of symbols used in the synchronizations; these are intro-

```
MODULE main
VAR gate1 : gate;
VAR gate2 : gate;

SYNC gate1, gate2 EVENTS open, open;
SYNC gate1, gate2 EVENTS close, close;

MODULE gate
VAR
  location : {closed, opening, opened, closing};
  timer : continuous;

EVENT open, close, tau;

INIT
  location = closed & timer = 0;

TRANS
  EVENT = open ->
    (location = closed & next(location) = opening & next(timer) = 0)

TRANS
  EVENT = close ->
    (location = opened & next(location) = closing & next(timer) = 0)

TRANS
  EVENT = tau ->
    ((location = opening & next(location) = opened) |
     (location = closing & next(location) = closed))

TRANS
  EVENT = tau -> (timer >= 10 & next(timer) = timer)

INVAR
  (location = opening -> timer <= 10) &
  (location = closing -> timer <= 10)

FLOW
  (location in {closed, opened)  -> der(timer) = 0) &
  (location = {opening, closing} -> der(timer) = 1)
```

Figure 9.1: Gate model written in HyDI.

duced with the keyword `EVENT`, which can also be used in the `TRANS` constraints as it was an input variable; intuitively, transitions are distinguished by the event which is being fired; for example, the transition from state s_1 to state s_2 is labelled with the event *closing*.

- *continuous* variables, a new type of variables declared with the keyword `continuous`; these are variables which are allowed to change in a timed transition and evolve according to some function continuous in time; for example, the variable *timer* is continuous and changes only during timed transitions; all other variables (Boolean, real, integer, etc.) are considered *discrete*, i.e., they change only during discrete instantaneous transitions.
- *flow* conditions, used to constrain the continuous evolution of continuous variables; the constraints are introduced with the keyword `FLOW` and may refer to the derivatives of continuous variables, denoted with `der`; for example, in the state s_2, the variable *timer* can only increase.

Intuitively, the system performs discrete and continuous transitions. In the former case, the whole system evolves as stated in the `TRANS` declarations. In the latter case, the discrete variables do not change, while the continuous variables change according to the `FLOW` conditions and to the elapsed time. For example, $s_2 = \langle state = closing, timer = 0\rangle$ moves to $s_3 = \langle state = closing, timer = 10\rangle$ with a continuous transition if $der(timer) = 10$ and the elapsed time is 10.

Processes

HyDI processes are instantiation of HyDI modules (in the example, `gate1` and `gate2` are processes). Differently from SMV processes, they can run both asynchronously or synchronize on shared events. Processes are declared in the *main* module of a HyDI model. They represent the components of a network whose topology is defined by the synchronizations. The network is not hierarchical in that there is no further asynchronous decomposition of a process, although the modules may contain synchronous instantiation of other modules.

Processes can share variables through the passage of parameters in the instantiation. However, they are limited to read the variables of other processes. This permits an easy identification of when the variables do not change even if the transitions are described with a generic relation (compared to a more restrictive functional description).

Synchronizations

Synchronizations specify if two events of two processes must happen at the same time. If two events are not synchronized, they interleave. Such synchronization is quite standard in automata theory and process algebra. It has been generalized with guards to restrict when the synchronization can happen. In the example, all the transitions of $gate_1$ labelled with *open* have to synchronize with a transition of $gate_2$ labelled with *open*.

In order to capture the semantics of some design languages, it is necessary to enrich the synchronization with further constraints that specify a particular policy scheduling the interaction of the processes. For this reason, it is possible to specify a *scheduler* in the main module of the HyDI model in terms of state variables, initial and transition conditions. These conditions may predicate over the events of the processes.

9.3.2 HyDI- syntax and semantics

In this section we present the *abstract* syntax and semantics of the HyDI language. Here we present the semantic of the language without shared variables and custom scheduler in the *main* module (See [CMT11b] for the details).

Abstract Syntax

A HyDI program is defined by a set of *processes* and by a set of synchronization constraints among the processes. Each process defines a (symbolic) hybrid automaton, while the synchronization constraints define a mapping among the labels of the automata to force a synchronization.

A process is defined with a HyDI *module*. A HyDI *module* is a tuple $\langle \text{PARAM}, \text{VAR}, \text{IVAR}, \text{INIT}, \text{TRANS}, \text{INVAR}, \text{FLOW} \rangle$, where:

- PARAM is a set P of formal parameters,
- VAR is a set of variable declarations defining a set of variables and, for each variable v, a type $\tau(v)$. The type $\tau(v)$ may be also *continuous*, to specify a continuous variable. Let X be the set of the continuous variables (the one declared of *continuous* type) declared in VAR and V be the set of the discrete variables declared in VAR,
- IVAR is a set of input variable declarations defining a set W of variables and for each variable w a type $\tau(w)$,
- INIT is a set of initial condition declarations defining a formula $Init$ over the variables $V \cup X \cup P$,
- TRANS is a set of transition condition declarations defining a formula $Trans$ over the variables $V \cup X \cup P \cup W \cup V' \cup X' \cup P'$,
- INVAR is a set of invariant condition declarations defining a formula $Invar$ over the variables $V \cup X \cup P$,
- FLOW is a set of flow condition declarations defining a formula $Flow$ over the variables $V \cup X \cup P \cup \dot{X} \cup \dot{P}$,

The VAR declaration may contain also module instantiation, i.e., variables whose type is in the form $\langle M, \beta \rangle$, where M is a module and β is a function that associates each formal parameter to an actual parameter. For every parameter $p \in P$, the actual parameter $\beta(p)$ must evaluate to the same type of p.

A HYDI program is composed of a set of module declarations and a *main* module. The *main* module declares the set of process instances $\mathcal{I}$ and constraints over the variables of the processes. In the following, we assume that the *main* module does not declare variables that are not instances and constraints different from the synchronization constraints. In [CMT11b] we define an extension of the language where the *main* module declares variables and arbitrary constraints.

A HYDI program P_{HYDI} is a tuple $\langle \mathcal{M}, main \rangle$ where:

- $\mathcal{M} = \{M_1, \dots, M_m\}$, such that each M_i is a module declaration $\langle \text{PARAM}_i, \text{VAR}_i, \text{IVAR}_i, \text{INIT}_i, \text{TRANS}_i, \text{INVAR}_i, \text{FLOW}_i \rangle$,
- $main = \langle \text{VAR}, \text{SYNC} \rangle$ is the *main* module such that:
  - VAR is a set of declarations defining the set of instances $\mathcal{I}$,
  - SYNC is a set if synchronization constraints. A synchronization in SYNC is a tuple $\langle i, j, a_i, a_j \rangle$, where $i, j \in \mathcal{I}$, $a_i \in \tau(i.\text{EVENT}_i)$ and $a_j \in \tau(j.\text{EVENT}_i)$ (See the requirement on EVENT_i for each instance $i \in \mathcal{I}$).
- for all the instances $i \in \mathcal{I} = \{i | i \in V \text{ and } \tau(i) = \langle M_i, \beta_i \rangle\}$:
  - $M_i = \langle \text{PARAM}_i, \text{VAR}_i, \text{IVAR}_i, \text{INIT}_i, \text{TRANS}_i, \text{INVAR}_i, \text{FLOW}_i \rangle \in \mathcal{M}$,
  - $\text{PARAM}_i = \emptyset$ is a set of empty parameters,
  - there exists an input variable $\text{EVENT}_i \in W_i$, such that $\tau(\text{EVENT}_i) = \{a_1, \dots, a_j\}$, such that $a_k \neq \text{S}$, for $1 \leq k \leq j$.

$\text{PARAM}_i = \emptyset$ means that there are no shared variables among the HYDI processes. In [CMT11b] we present the extended semantic with shared variables[4]. A synchronization $\langle i, j, a_i, a_j \rangle$ in SYNC enforces the instances i and j to perform a transition labelled with the event a_i and a_j at the same time.

HyDI- Semantics

In the following, we define the semantic of a HyDI program $P_{\text{HyDI}} = \langle \mathcal{M}, main \rangle$ as a network of hybrid automata $\mathcal{N} = ||_{i \in \mathcal{I}} H_i$, where each H_i is obtained from one instance $i \in \mathcal{I}$.

Each instance $i \in \mathcal{I}$ may declare sub-instances of another module type. For example, the i-th process instance may declare an instance of type $\langle M_j, \beta_j \rangle$. Then, also M_j may declare instances of some module type, thus allowing for a hierarchical definition of the program. We assume that the program does not have circular dependencies among modules (See [McM93] for details).

We define the function ϕ that, given a module definition M, returns the module definition M' where all the instances of M have been instantiated. Given a module definition $M = \langle \text{PARAM}, \text{VAR}, \text{IVAR}, \text{INIT}, \text{TRANS}, \text{INVAR}, \text{FLOW} \rangle$, $\phi(M) = \langle \text{PARAM}', \text{VAR}', \text{IVAR}', \text{INIT}', \text{TRANS}', \text{INVAR}', \text{FLOW}' \rangle$ is defined as follows. If M does not define any instance (i.e. for each $v \in V$, $\tau(v) \neq \langle M_j, \beta_j \rangle$, for some module definition M_j), then $\phi(M) = M$.
Otherwise, let $\mathcal{I}_M$ be the set of instances of M. For each $i \in \mathcal{I}_M$ with type $\tau(i) = \langle M_i, \beta_i \rangle$, we consider the type $\langle \phi(M_i), \beta_i \rangle$, obtained recursively applying ϕ to the module type of the instance i (i.e. by recursively applying ϕ we assume that the module type of each i is already instantiated). Let $\phi(M_i) = \langle \text{PARAM}_i, \text{VAR}_i, \text{IVAR}_i, \text{INIT}_i, \text{TRANS}_i, \text{INVAR}_i, \text{FLOW}_i \rangle$. $\phi(M)$ is defined as follows:

[4]When there are shared variables among processes we cannot define anymore the local-time semantic encoding. Moreover, the language restricts shared variables to be read-only (i.e. a process can never write a variable of another process).

- $\textsc{param}' = \textsc{param}$.
- $\textsc{var}' = (\textsc{var} \setminus \mathcal{I}_M) \cup \bigcup_{i \in \mathcal{I}_M} (i.\textsc{var}_i)$,
  where with $i.\textsc{var}_i$ we denote the set of declarations obtained copying all the declarations $\textsc{var}_i$ renaming each variable v with $i.v$,
- $\textsc{ivar}' = \textsc{ivar} \cup \bigcup_{i \in \mathcal{I}_M} (i.\textsc{ivar}_i)$,
- $\textsc{init} = \textsc{init} \cup \bigcup_{i \in \mathcal{I}_M} \textsc{init}'_i$,
  where $\textsc{init}'_i$ is obtained from $\textsc{init}_i$ renaming each variable $v \in X_i \cup V_i \cup W_i$ with $i.v$ and each variable $p \in P_i$ with $i.\beta(p)$,
- $\textsc{trans}' = \textsc{trans}' \cup \bigcup_{i \in \mathcal{I}_M} \textsc{trans}'_i$,
  where $\textsc{trans}'_i$ is obtained from $\textsc{trans}_i$ renaming each variable $v \in X_i \cup V_i \cup W_i$ with $i.v$ and each variable $p \in P_i$ with $i.\beta(p)$, and each variable $v' \in X'_i \cup V'_i \cup W'_i$ with $i.v$ and each variable $p' \in P'_i$ with $\beta(p)'$,
- $\textsc{invar}' = \textsc{invar} \cup \bigcup_{i \in \mathcal{I}_M} \textsc{invar}'_i$
  where $\textsc{invar}'_i$ is obtained from $\textsc{invar}_i$ renaming each variable $v \in X_i \cup V_i \cup W_i$ with $i.v$ and each variable $p \in P_i$ with $i.\beta(p)$,
- $\textsc{flow}' = \textsc{flow} \cup \bigcup_{i \in \mathcal{I}_M} \textsc{flow}'_i$
  where $\textsc{flow}'_i$ is obtained from $\textsc{flow}_i$ renaming each variable $v \in X_i \cup V_i \cup W_i$ with $i.v$, each variable $p \in P_i$ with $i.\beta(p)$ and replacing the occurrences of $\dot{v}$ with $\dot{i.v}$.

For each instance $i \in \mathcal{I}$ of a program $\langle \mathcal{M}, main \rangle$ we assume that it has been instantiated (i.e. we consider the type $\langle \phi(M_i), \beta_i \rangle$ instead of the type $\tau(\langle M_i, \beta_i \rangle)$. Thus, we assume that $\textsc{var}_i$ of M_i does not contain instance declarations anymore.

The synchronization constraints are declared between couples of processes. We define the transitive synchronization relation $\textsc{sync}^*$ from $\textsc{sync}$, which represents the set of all the possible pairwise synchronization of the network. The tuple $\langle i, j, a_i, a_j\rangle$ is in SYNC* iff there exists a sequence of instances $l_1, l_2, \ldots, l_n$ such that $\langle l_k, l_{k+1}, a_{l_k}, a_{l_{k+1}}\rangle \in \textsc{sync}$ for $1 \leq k \leq n$, $i = l_1$ and $j = l_n$.

Note that each process in the program declares events with different domains. Then, the synchronization constraint $\langle i, j, a_i, a_j\rangle$ forces that every time the process i move with event a_i also the process j moves with event a_j. In the hybrid automata network formalism automata move on the same label. Thus, we define a renaming function for the event values, ρ. The function $\rho(i, a_i)$ returns the event value a_i if the event a_i is not involved in any synchronization of the process i. Otherwise, it returns the same values for all the processes and events involved in the same synchronization (e.g. if $\textsc{sync}^* = \{\langle i, j, a_i, a_j\rangle, \langle j, i, a_j, a_i\rangle, \langle i, k, a_i, a_k\rangle, \langle k, i, a_k, a_i\rangle, \langle j, k, a_j, a_k\rangle,$ $\langle k, j, a_k, a_j\rangle\}$, then $\rho(i, a_i) = \rho(j, a_j) = \rho(k, a_k)$). Moreover, we assume that this value is unique for each synchronization and does not clash with local event values.

Now, we provide the semantic of the HYDI program $P_{\text{HYDI}} = \langle \mathcal{M}, main\rangle$ as a network of hybrid automata $\mathcal{N} = ||_{i\in\mathcal{I}} H_i$. For each $i \in \mathcal{I}$ with type $\tau(\langle M\rangle, \beta)$, we define H_i as the hybrid automaton $H_i = \langle V_{H_i}, X_{H_i}, \varepsilon_{H_i}, Init_{Hi},$ $Invar_{H_i}, Trans_{H_i}, Flow_{H_i}\rangle$ such that:

- $V_{H_i} = V_i \cup (\mathcal{W}_i \setminus \{\text{EVENT}_i\})$,
- $X_{H_i} = X_i$,
- ε_{H_i} is such that its domain $Dom(\varepsilon_{H_i}) = \{\rho(i, a_i) \mid a_i \in \tau(i.\text{EVENT}_i)\}$,
- $Init_{H_i} = Init_i$,
- $Invar_{H_i} = Invar_i$,
- $Trans_{H_i}$ is obtained substituting in $Trans_i$ each predicate $\text{EVENT}_i \bowtie a$, where $\bowtie \in \{=, \neq\}$, with $\varepsilon_{H_i} \bowtie \rho(i, a)$,

- $Flow_{H_i} = Flow_i$.

9.4 Related work

There exists several tools and languages to analyze and verify Hybrid Systems.

Several tools verify invariant properties computing the set of the reachable states. Uppaal [BLL+95] model checks a subset of TCTL (Timed Computation Tree Logic) [ACD90] properties for timed automata. It computes the set of the reachable states using specialized data structures (Difference Bounded Matrix). The reachability is explicit in the discrete states of the automata. The tool does not handle hybrid systems, arbitrary data types and LTL properties. Moreover, Uppaal does not allow the user to model parametric designs. HyTech [HHWT97] is a model checker for linear hybrid automata, which represents the continuous part of the reachable states using polyhedra. Phaver [Fre08] and SpaceEx [FGD+11a] model affine continuous dynamics with inputs. They check invariant properties computing an approximation of the set of the reachable states using different techniques (Polyhedra and support functions). Other model checkers, HSolver [RS07], d/dt [ADM02] and Ariadne [BBC+14], verify invariants of non-linear hybrid systems.

KeYmaera [PQ08] is a theorem prover for hybrid systems. It can handle non-linear hybrid systems, with symbolic parameters and also with an unbounded number of components. It is based on deductive verification techniques.

HybridSAL [Tiw12] is very similar to HyComp. The tool encodes linear hybrid systems as infinite-state transition systems, which can be verified using the SAL [dMOR+04] model checker. HybridSAL also implements other abstraction techniques (e.g. See [ZST12]), but it does not implement the quantifier free encoding for polynomial hybrid systems. The tool cannot prove LTL properties, it does not provide verification algorithms that exploit the hybrid automata network, and does not implement the efficient verification algorithms for invariant properties (e.g. IC3).

Finally, Atmoc verifies invariant, LTL [KJN12a] and MITL [KJN13] properties for symbolic timed automata.

Comparing the existing input languages for hybrid automata, the one adopted by the explicit verification tools (Uppaal, HyTech, Phaver, SpaceEx, d/dt, HSolver, Ariadne) use an explicit representation of the discrete locations and transitions, which are explicitly enumerated in the model. For this reason they cannot specify flow and invariant conditions for a set of locations. Moreover, these tools do not allow rich types for discrete variables, such as Boolean, Enumeration, Word, Unbounded Integer and Unbounded Real. HyTech assumes the parallel composition of all automata, which synchronize on labels with the same name, while in Phaver the user can specify which automata synchronize. SpaceEx has a compositional language based on the Hybrid I/O automata formalism that enables compositional verification.

Other languages proposed to model hybrid systems focus on hierarchical specifications (e.g. Charon [AGH$^+$00]) or compositionality (e.g. Masaccio [Hen00]). The *Hybrid Systems Interchange Format* (Hsif) [tea02] and the *Common Interchange Format* (Cif) [vBRSR07] were proposed as standards to represent and interchange models of Hybrid Systems. Hsif does not allow hierarchical state machines, while Cif allows to write rich constraints (DAE, Differential Algebraic Equations), which mix also variables and derivatives.

With respect to languages that use a fully symbolic representation, HybridSAL allow for a hierarchical definition of synchronous and asynchronous systems. In the language it is not easy to to express the synchronization of asynchronous components via discrete interaction, as in the hybrid automata case. TSMV is the language used by ATMOC. It extends the language of NuSMV adding clocks and reset of continuous variables. The language only consider synchronous systems (i.e. it does not consider networks of automata) and is limited to timed automata. The language HLANG [FHSW07] is very close to HYDI and was developed in the project area H of AVACS as intermediate input language for several verification tools (e.g. HYSAT [FH07], FOMC [DDD$^+$12]). The language employs a symbolic representation for hybrid automata and enables very expressive constraints for continuous variables, also affine and non-linear. The language allows to express the composition of hybrid automata. However, the automata communicates through shared variables and automata are composed interleaving their transitions. Thus, there is no native support for event-based synchronizations.

Hybrid programs [BS10, Pla10] are similar to programs for discrete systems, but they add the description of the continuous evolution. [BS10] extends the synchronous language Quartz [Sch09a] with continuous variables. The user specifies in the program when the continuous evolution can happen. The semantic is such that the discrete statements in the program are instantaneous, while the statements over continuous variables allow the elapse of time. The continuous evolution is specified using ODE (Ordinary Differential Equations). Assignments in the program are expressed in a functional form, thus they are less expressive than the relational representation of HYDI. Two blocks of statements can run in parallel (synchronous composition), while there is no support for asynchronous composition. These hybrid programs are then translated into a monolithic extended finite state machine. Also the hybrid programs used in KEYMAERA, defined by Platzer in [Pla10], allow rich constraints over continuous variables, sequences, loops and non-deterministic choices of statements. However, they miss the possibility of expressing the parallel execution of programs. Hybrid programs are very different in nature from the representation used in HyDi.

Chapter 10

Experimental Results

Note. The material presented in this chapter has already been presented in [CMT12, CMT13b, MCTT13, BCL+10a, CMT11a, CMT11c, CMT13c, CGMT13]

In this chapter we report the experimental evaluations performed to assess the efficiency of the techniques described in the thesis.

All the technique proposed in the thesis, except for time-aware relational abstraction, have been implemented in the HyComp [hyc] and in the nuXmv [nux] tools.

The nuXmv tool extends the NuSMV [CCG+02] model checker in two main directions. The first direction is the implementation of efficient model checking algorithms for finite-state systems, like IC3 and k-liveness [CS12b]. Then, nuXmv extends the input language of NuSMV adding infinite-state types (e.g. real and integers), an interface to the MathSAT SMT solver [CGSS13] and the implementation of several algorithms for the verification of infinite state systems. nuXmv implements the implicit abstraction technique presented in Section 6.3 and the parameter synthesis algorithm of Chapter 8.

HyComp implements the encoding techniques presented in Chapters 3,4 and Chapter 5, the bounded model checking algorithm based on shallow synchronization 6.2 and all the scenario verification techniques 7. Note that HyComp may exploit all the algorithms for infinite-state systems implemented by nuXmv, since the tool encodes a network of hybrid automata in a symbolic transition systems, either in the internal representation of nuXmv or as a file in the concrete syntax of its input language.

Instead, for the evaluation of the time-aware relational abstraction we implemented the technique in the HybridSAL tool [Tiw12]. Then, we used the K-induction implementation of the SAL model checker [dMOR+04], which is based on the SMT solver Yices[1]. Both tools are freely available

[1]`http://yices.csl.sri.com/`

from SRI International.

The Chapter mainly follows the structure of the thesis and report, for each technique, the related experimental evaluation. Before presenting the experimental results, in Section 10.1 we describe the set of benchmarks that we used in the different experimental evaluations. In details, in Section 10.2.1 we report the experimental evaluation related to the different encodings of hybrid automata and to relational abstraction. Then, in Section 10.3 we present the experiments related to reachability analysis, both for falsifying a property with Bounded Model Checking and for proving it using implicit abstraction. In Section 10.4 we present a thorough experimental evaluation on the scenario verification problem. Finally, Section 10.5 presents the effectiveness of the parameter synthesis algorithm based on IC3.

10.1 Benchmarks

In this section we describe the different benchmarks used in the experimental evaluation. When the experiments have been performed with HyComp, we used HyDI as language to describe the input models. In the case of time-aware relational abstraction, the benchmarks where modeled in HybridSAL.

***ETCS* [HEFT08].** Industrial case study of the braking control system of trains. The example is inspired by the European Train Control System (ETCS) specification that controls the movement of trains on a track divided into sections. We consider the version of [HEFT08] where trains move with a uniformly accelerated motion, and a second version where the dynamic is approximated with linear constraints.

Bouncing Ball. The benchmark models a ball that fall vertically and bounce on a surface. We used four variants: 1. a ball moving vertically in one dimension and bouncing on a plain floor, 2. a two-dimensional variant, where the ball falls and also moves horizontally with constant horizontal speed, 3. a ball always moving in two dimension as before, but bouncing on a hill (vertical parabola) and 4. a ball bouncing on a slope (horizontal parabola).

Ballistic. The benchmark models an object that flies above an obstacle (a hill) keeping below a flat ceiling.

Fischer protocol benchmarks. We considered different versions of the Fischer mutual exclusion protocol that regulates the access of processes to a critical section:

- *Fischer timed*: standard version of the Fischer protocol with clock variables (i.e. timed automata).
- *Star-shape Fischer* [Wan05]: hybrid version of the protocol where clocks are replaced by continuous variables with rectangular dynamic (i.e. $\dot{x} \in [lower, upper]$), and where the shared variable is modeled with an additional automaton. For parameter synthesis we consider a version with a free parameter, while in both reachability and scenario verification we fixed the parameter to a constant.
- *Ring-shape Fischer*: this variant always considers rectangular dynamics but contains a ring of processes, where each process shares a variable with its left and right neighbors; the variables are used to access the critical sections in mutual exclusion with the neighbors.

Simple Ring. This example is a simple ring of processes where each process only communicates with its left and right neighbors.

Motorcycle. This example is inspired by the automated highway system from [JKWC07]. This system models a sequence of n motorcycles. Every motorcycle needs to preserve its relative position in the sequence by synchronizing shared labels with neighbors to accelerate/decelerate. Each motorcycle i needs to wait the signal from the previous one to move, and it needs to keep the sequence during the parade by synchronizing shared labels with neighbors. Once two motorcycles i and $i+1$ find the distance between them are not safe, they will use share label $close\_i\_i+1$ to jump to accelerate and decelerate mode from cruise mode and come back to cruise mode by share label $not\_close\_i\_i+1$.

FDDI Protocol. This example is a ring topology model based on the system in [ZLZZ03]. It is a set of standards for data transmission on fiber optic lines in a LAN. Each component in the system waits for the signal of previous one to transmit data. For parameter synthesis, we used the version of [BLN03] with unbounded parameters.

***Nuclear Reactor* [Wan05].** The benchmark models the control system of a nuclear reactor with n rods, and uses these rods to absorb neutrons one by one. Each rod that has just been moved out must stay out of the water and cool for several time units.

Multi-Frequency. This example models a global controller that periodically samples the value of a variable from n local controllers. Each local controller reads the value of a sensors with a higher frequency than the global controller. Moreover, all the controllers have a different phase (i.e.

they start the sampling cycle with a different time offset, which shifts the cycle by a constant amount of time).

Audio Protocol [**HH94**]. The benchmark models a protocol that transmits an arbitrary-length bit sequence from a sender to a receiver based on the timing-based Manchester encoding. The protocol relies on division of the elapsed time in slots. Every slot corresponds to a bit. The sender transmits a signal *up* in the slots corresponding to bits with value 1 (thus, a slot without signals correspond to bit 0). The protocol is robust to bounded errors in the timers used by the sender and receiver.

Distributed Controller [**HH94**]. The benchmark models the interactions of n sensors with a preemptive scheduler and a controller. The sensors interact with the scheduler to access the controller, which must read data from all the sensors in order to complete the computation.

Electronic Height Control System [**MS00**]. The benchmark is an industrial case study of the system that controls the height of a car's chassis. It consists of a controller that changes the height of the chassis (turning on or off a compressor and opening a valve) in order to keep its height in a desired interval. A timer tells the controller when sampling the height from a filter, while disturbances that changes the height of the vehicle are modeled by the environment.

CSMA/CD [**Wan05**]. The benchmark is a model of the CSMA/CD protocol.

Schedulability [**CPR08**]. The benchmark models the schedulability problem of several tasks with a network of timed automata.

Train Gate Controller **[Wan05].** This is a parametric and scalable model of the train-gate-controller benchmark. The model guarantees that bars at a railroad crossing are closed when a train passes. The model has an unknown parameter on the amount of time needed to lower the bar.

PID controller. We considered several version of a Proportional-Integral-Derivative (PID) controller taken from a MATLAB tutorial [2]. The controller is continuous and reads the error signal specified by the output signal of the plant and a reference signal. The output of the controller is given as input to the plant. The goal of the controller is to minimize the error signal (i.e. the difference between the signal of the plant and the desired behavior for the plant).

In our benchmark, the plant is a simple mass, spring and damper system. The modeling equation of the mass, spring, and damper system (plant) is

$$M\ddot{x} + b\dot{x} + kx \;=\; F$$

where $M = 1kg$, $b = 10Ns/m$, $k = 20N/m$ are the given parameters of the plant, and F is the (controllable) force. Suppose the goal is to make the plant reach a steady state where $x = 1$ with the some requirements on the overshoot and rise time (that we will precisely specify later). Suppose the desired trajectory $r(t)$ for reaching the steady state $x = 1$ is a step function: at time $t = 0$, we want the system to go from its initial state (say, $x = 0$) to its steady state $x = 1$; that is, $r(t) = 0$ for $t < 0$ and $r(t) = 1$ for $t \geq 0$.

Let us assume that we are given a PID controller that has gains $K_p = 350, K_i = 300, K_d = 50$. The equation describing the composed controller and plant system is

$$M\ddot{x} + b\dot{x} + kx \;=\; K_d(r \dot{-} x) + K_p(r - x) + K_i \int (r - x)$$

[2] `ctms.engin.umich.edu/CTMS/`

Note that $r - x$ is the error in tracking the desired trajectory r. Substituting the parameters given above in this equation, we get the following state-space model of the controller and the plant subsystem. (Since r is not differentiable at $t = 0$, we have used $\dot{r} = 0$ here).

$$\begin{aligned} \frac{dxint}{dt} &= x \\ \frac{dx}{dt} &= xder \\ \frac{dxder}{dt} &= -60 * xder - 370 * x - 300 * xint + 350 + 300 * \mathtt{t} \\ \frac{d\mathtt{t}}{dt} &= 1 \end{aligned}$$

where x, $xder$ (denoting $\dot{x}$), $xint$ (denoting $\int x$), and t are the four state variables.

The benchmark was modeled in HybridSAL.

Active suspension. The model was derived from the paper [FB02]. The 1/4-car active suspension model consists of 5 state variables. There are four modes in the hybrid automaton. The modes arise from gain scheduling – essentially different parameters are used in the controller in different regions of the state space. The benchmark was modeled in HybridSAL.

10.2 Encodings

10.2.1 Quantifier-free encoding

Evaluation Settings

We performed several experimental evaluation on benchmarks from the class polynomial hybrid automata (we considered the *ETCS*, the *Bouncing Ball* and the *Ballistic* benchmarks). First, we evaluated the feasibility of using quantifier-elimination procedures to remove the quantifier from the encoding. Then, we performed Bounded Model Checking on several instances to asses the feasibility of the quantifier-free encoding and to compare the performances with the encoding obtained using quantifier-elimination procedures. Finally, we tried to apply our encoding to a simple inductive property.

The encoding in the polynomial case is automatic and it is implemented in HyComp. The Bounded Model Checking is non-incremental and uses iSAT[3] as backend to solve the resulting satisfiability queries. In the BMC settings we usually perform a search where we check if the target is violated for an increasing path length. Then, the removal of the quantifiers requires more continuous transitions, thus increasing the size of the formula passed to the solver. We considered a "layered" approach, where we first reach the target in an over-approximation of the system, where invariants are not guaranteed to hold, and then we check if there exists a path that reaches the target and for which invariants hold.

All the benchmarks are publicly available at `http://es.fbk.eu/people/mover/tests/FMSD_FMCADSI/`. The archive also include the non-linear benchmarks described in Section 3.2.4.

Results

We evaluated the alternative use of quantifier elimination procedures, within their range of applicability, i.e. polynomial hybrid automata. We experimented with Cylindrical Algebraic Decomposition (CAD) (using QepCAD[4]) and Virtual Substitution (using RedLog[5]). Table 10.1 reports, for each polynomial benchmark, the time needed to obtain a quantifier free formula of the invariants using QepCAD and RedLog. The Virtual Substitution approach of RedLog can only handle formulas quantified

[3]http://isat.gforge.avacs.org/
[4]http://www.usna.edu/cs/ qepcad/B/QEPCAD.html
[5]http://redlog.dolzmann.de/

	# vars	Max degree	RedLog	QepCAD
etcs_braking	4	2	0.14	0.05
ball_1d_plain	4	2	0.10	0.03
ball_2d_plain	4	2	0.10	0.03
ball_2d_hill	5	2	0.15	T.O. > 3600.00
ball_2d_slope	5	4	N.A.	T.O. > 3600.00
simple_ballistics	5	4	N.A.	T.O. > 3600.00

Table 10.1: Results of applying quantifier elimination to the polynomial benchmarks (max degree is the maximum degree of the quantified variable,T.O.is a time out of 3600 seconds, N.A. means not applicable).

over a quadratic variable. QepCAD is slightly more general, but de facto less useful: the results highlight the dramatic computational complexity of the procedure (e.g. *ball_2d_hill*, with 5 variables, times out in one hour). Thus, the quantifier elimination approach cannot even handle the polynomial benchmarks ballistic and *ball_2d_slope* (in addition to the benchmarks with transcendental functions).

We used the bounded model checking functionalities enabled by our approach to validate the various models and to evaluate the performance of the invariant encoding. For each model we generated different reachability properties which are falsified by traces with an increasing length. We evaluated the encoding of the invariant by comparing the time needed to find these traces with BMC. When quantifier elimination was able to produce a result, we also compared it with our approach using the same SMT-based technique, in order to evaluate the overhead caused by the splitting. The results are shown in Table 10.2. The encoding time of our approach is instantaneous in all cases. In the cases where quantifier elimination is feasible, the resulting encoding may induce traces with a smaller number of steps, because timed transitions must not be split. This happens for the *ball_1d_plain* and the *ball_2d_hill* benchmarks. The reduced

	quantifier-free encoding	qelim (qepcad)	qelim (redlog)
etcs_braking	66.75 / 17	161.52 / 17	168.16 / 17
ball_1d_plain.01	0.05 / 2	0.05 / 2	0.03 / 2
ball_1d_plain.02	25.50 / 6	0.09 / 4	0.06 / 4
ball_1d_plain.03	31.43 / 10	0.28 / 6	0.40 / 6
ball_1d_plain.04	36.23 / 14	0.46 / 8	0.65 / 8
ball_1d_plain.05	151.41 / 18	1.27 / 10	1.51 / 10
ball_2d_plain.01	0.08 / 2	0.18 / 2	0.28 / 2
ball_2d_plain.02	4.20 / 6	3.14 / 6	3.64 / 6
ball_2d_plain.03	16.04 / 10	15.90 / 10	62.64 / 10
ball_2d_hill.01	1.30 / 4	na / na	0.94 / 2
ball_2d_hill.02	118.67 / 8	na / na	15.36 / 4
ball_2d_slope.01	to / na	na / na	na / na
simple_ballistics	8.31 / 1	na / na	na / na

Table 10.2: Results (running time / path length) of BMC with the different encodings.

number of steps also reduces the time needed to generate the trace.

Our approach was also able to prove a simple invariant on the ballistics example, that was beyond the applicability of SMT-based techniques. We chose as obstacle a circle shape with center in $(c, 0)$ and radius r. If the ceiling level is less than r, the object cannot clearly pass. This has been proved with HyComp and iSAT. Ignoring the invariant along the timed transitions (keeping it only on the discrete points) allows for spurious traces that forbid the inductive proof. Note that this small example is beyond the applicability of quantifier elimination (see Table 10.1).

Some remarks are in order. Our approach strongly depends on the availability of SMT solvers for quantifier-free theories of nonlinear arithmetic, to solve the formulas resulting from our SMT-based verification engines. To this end, we tried to use all the available solvers for nonlinear arithmetic: Z3[6], SMT-RAT[7], CVC3[8], miniSMT[9], RAHD[10], hydlogic[11], dReal[12]. and iSAT[13]. Z3 and SMT-RAT implement two complete decision procedures for the non-linear arithmetic over reals. Currently, neither solver integrate a layering with the linear arithmetic solver: in this case all the linear arithmetic constraints are handled using the non-linear solver, thus resulting in an inefficient approach. This is the case for our BMC case studies, which have a significant part of linear constraints. Instead, CVC3 and miniSMT implement an incomplete decision procedure for non-linear arithmetic (and miniSMT is tailored only to check satisfiable formulas). As a result, these solvers turned out to return "unknown" on most of the queries generated from our benchmarks. The hydlogic system turned out to be immature, while RAHD exports functionalities that are closer to a theory solver than a full SMT solver, requiring an explicit treatment of disjunctions. iSAT and dReal differ from the other solvers, since they can also provide approximate solutions. dReal returns an unsatisfiable answer or a satisfiable answer if the formula is satisfiable under a bounded numerical perturbations. iSAT may return "unknown" exposing the results of interval constraints propagation: it produces the intervals found in the search, if these are below a user-defined threshold, as a candidate solution. In many practical cases, this is not spurious, and represents a satisfying assignment of the formula.

Overall, despite some recent progress, our experience has shown that the field still requires additional research to deliver what our approach requires, both in terms of completeness, and performance. However, we argue that

[6]http://research.microsoft.com/en-us/um/redmond/projects/z3/
[7]http://smtrat.sourceforge.net/
[8]http://cs.nyu.edu/acsys/cvc3/
[9]http://cl-informatik.uibk.ac.at/software/minismt/
[10]http://homepages.inf.ed.ac.uk/s0793114/rahd/
[11]http://code.google.com/p/hydlogic/
[12]http://www.cs.cmu.edu/ sicung/dReal/
[13]http://isat.gforge.avacs.org/

our method is valuable regardless of the current status of SMT for nonlinear arithmetic. First, we proposed a solution to a problem that was a show-stopper for SMT-based verification. In fact, we are now able to solve some benchmarks that cannot be solved by overapproximation, just forgetting about the quantified invariants. Second, we are hopeful that the field of SMT can deliver quick progress in quantifier-free nonlinear arithmetic. In fact, the development of SMT solving for non-linear arithmetic has been influenced by benchmarks from other domains (e.g. most of the SMT-LIB benchmarks in NRA are from the software domain). To this extent, we generated and submitted to the SMT-LIB a vast number of benchmarks, that will trigger additional research in practically relevant directions.

10.2.2 Time-aware relational abstraction

Evaluation Settings

We extended the HybridSAL tool to compute also the time-aware relational abstractions. The computation of the abstraction has three parameters:

- l, m: determine the first, $(-\infty, e^{-l}]$, and the last, $[e^{m}, \infty)$, intervals for the piecewise approximations of logarithms (See Equations 5.9 for details). Recall from Section 5.3.3 that we guarantee a fixed error γ to compute the logarithm $ln(x)$ for the unbound intervals of $(-\infty, e^{-l}]$, $[e^{-l}, e^{-l+1}]$, $\ldots$, $[e^{m-1}, e^{m}]$, $[e^{m}, \infty)$, while the unbounded intervals only guarantee soundness.

- n: determine the number of complete cycles that lie between the initial and final state (See Equation 5.13 for details). For values greater than n we guarantee soundness, but the abstraction is less precise.

First, note that a bigger value of all three the parameters results in a more fine-grained approximation. Then, note that in the current implementation the parameters of the abstraction are the same for the whole abstraction: this means that we do not specify different parameters for different locations of the systems or for different eigenvalue and eigenvector pairs or for different logarithm functions. In principle, the refinement could be more fine-grained to achieve different precision.

HybridSAL generates abstractions in the language of the SAL model checker. We used SAL and its implementation of k-induction to prove safety properties on two case studies, the PID controller and the active suspension benchmark.

We consider several versions of the PID controller: a PD controller, a PI controller and a PID controller. Dropping one of the component of the controller (the proportional, P, or the integral, I, or the derivative, D) changes the behavior of the controller. The intuition is that the proportional component takes into account the current error, the integral component accumulates the past errors and the derivative component predicts the future errors. For each of the three controllers, we also consider two variants: one in which the integral term goes through a saturation block (suffixed with "Sat") and one in which there is no such saturation block. (In PD, the coefficient of the integral term is zero, but the integral is still computed and hence, saturated in PDSat.) For the PID controller, we wish to check the following rise time requirement: after t=0.5 units, x reaches within 90% of its steady-state value. We check a stronger variant of this requirement, namely

$$\mathtt{G}(\mathtt{t} > 0.5 \Rightarrow x \geq 0.9 \wedge x \leq 1.1)$$

which says that it is always true that whenever time is greater than 0.5, then x is in the $[0.9, 1.1]$ interval.

For the active suspension benchmark, we wish to prove that the deflection of the suspension always remains within a safe interval.

We evaluate our approach trying to prove the properties on both the case studies using the time-unaware and the time-aware relational abstraction with different precisions.

Results

In Table 10.3, we present the results of analyzing the rise-time requirements for different PID controllers. We analyze the six system models using time-agnostic relational abstraction, and using time-aware relational abstraction. The analysis with time-aware relational abstraction is carried out with four different settings of the three parameters (l, m, n).

We note that we cannot prove the rise-time requirement for any of the system models using only time-agnostic relational abstraction. Using time-aware abstraction, we also cannot prove the rise-time requirement for any of the models unless we pick $l \geq 3$. Note that the PD and PID systems (and their saturated counterparts) satisfy the desired rise-time requirement, **whereas the PI system does not**.

Note that we perform bounded model checking on the computed time-aware abstractions. In general, failure to find a counter-example in a bounded run does not imply the validity of the property. But, for single locations hybrid systems (such as PD, PI and PID), if there is no depth 1 counter example, then the property is valid. This is because (time-agnostic and time-aware) relational abstractions over-approximate unbounded time reach sets (for each mode). For hybrid systems with multiple locations, a proof using k-induction is required to prove a property.

We applied the time-aware relational abstraction technique to verifying bounded deflection in the active suspension model. Time-agnostic relational abstraction is unable to verify this safety property; it always produces a spurious counterexample. However, the time-aware abstraction constructed using parameter values $l = 2, m = 2, n = 2$, was sufficient to

	Time-Unaware Abstraction			**Time-Aware Abstraction**					
Model	HybridSAL *Time*	SAL *Time*	*Result*	l	m	n	HybridSAL *Time*	SAL *Time*	*Result*
PD	0.6	0.4	U	2	0	0	1.1	0.2	U
				3	0	0	0.4	0.2	U
				0	2	0	0.4	0.2	U
				4	2	2	0.6	0.6	P
PI	0.4	0.1	U	2	0	0	0.6	0.6	U
				3	0	0	0.4	0.2	U
				0	2	0	1.2	0.6	U
				4	2	2	0.5	0.2	U
PID	1.1	0.1	U	2	0	0	0.4	0.2	U
				3	0	0	0.4	0.2	P
				0	2	0	0.8	0.6	U
				4	2	2	1.2	0.8	P
PDSat	1.5	0.4	U	2	0	0	1.5	0.7	U
				3	0	0	0.5	0.2	U
				0	2	0	0.8	0.7	U
				4	2	2	0.8	0.7	P
PISat	0.6	0.2	U	2	0	0	1.2	0.3	U
				3	0	0	0.6	0.3	U
				0	2	0	0.9	1.1	U
				4	2	2	0.6	0.5	U
PIDSat	1.5	0.4	U	2	0	0	1.2	1.1	U
				3	0	0	0.6	0.3	P
				0	2	0	0.8	0.35	U
				4	2	2	1.3	1.6	P

Table 10.3: Results on verifying feedback PD, PI and PID controllers, with and without saturation, using different parameters for the time-aware relational abstraction. Model names with suffix “Sat” are versions that have saturation applied on the integral term. “HybridSAL time” is the time took by HybridSAL to compute the abstraction, “SAL time” is the time took by SAL to run k-induction at depth 1, l, m, n are the parameters of the time-aware relational abstraction. “Result” may be not-proved (U), if we find a counterexample, or proved (P) if k-induction proves the property.

show that there were no counterexamples up to depth 4.

10.3 Reachability

10.3.1 Shallow synchronization

Evaluation Settings

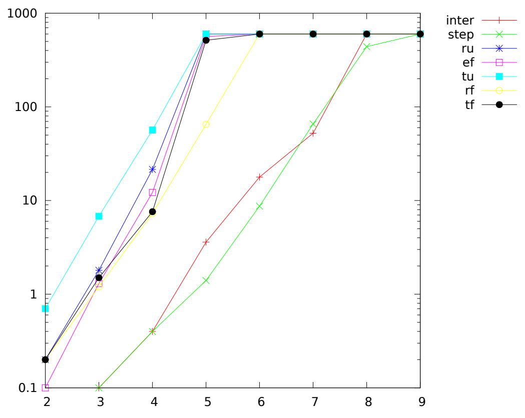
In this section we evaluate the effectiveness of the different encodings presented in Section 6.2.2. With respect to the different options of the encoding, we use the following notation:

- `e` for the enumerative encoding,
- `r` for the local reasoning encoding,
- `t` for the local reasoning with uninterpreted functions encoding.

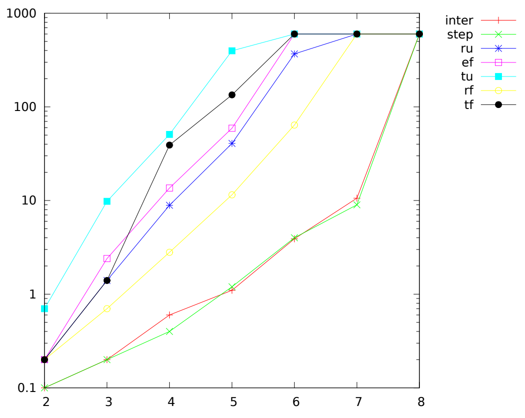
With regards to the incrementality, when we use local reasoning, we can add the synchronization constraints during the unrolling (denoted with `u`) or add them after the unrolling (denoted with `f`). Overall, we have 5 different configurations: `ru`, `rf`, `ef`, `tu`, `tf` (e.g., `ru` means using local encoding with the constraints added during the unrolling).

To evaluate the approach, we use the following benchmarks: *Simple Ring*, *Star-shape Fischer*, *Ring-shape Fischer*, *ETCS*, *Motorcycle*, *FDDI Protocol*, *Nuclear Reactor*, *Multi-Frequency*.

We check reachability problems comparing the encodings based on the global time semantic with and without the step semantic optimization and all the different options for the shallow synchronization. We compared the results only on reachable instances. For unreachable cases, since we are using a BMC approach, the results strongly depend on the fixed bound, but the meaning of the bound depends on the semantics: for the interleaving, it represents the total number of local and global steps; for the shallow



(a : *Star-shape Fischer*)



(b : *Nuclear Reactor*)

Figure 10.1: Scalable plots showing the run times (seconds on the y axes) to solve a problem with an increasing number of processes (on the x axes). For *Star-shape Fischer* and *Nuclear Reactor* length of a local run depends on the number of processes.

synchronization, it represents the maximum bound of a local run. Thus, any bound would be unfair for either semantics. Nevertheless, note that all algorithms check the unreachability of the target for path lengths smaller than the final one. So, the performance does not depend on the chance of finding the right path. We ran the experiments on a Red Hat 4.1.2 machine, with Intel(R) Core(TM)2 Quad CPU 2.66*4, and 4GB of RAM with a time out of 600 seconds.

Results

The results of the comparison are shown in Figures 10.1 and 10.2, where the time to solve the reachability problem is plotted in log scale against the number of automata in the network. Each line corresponds to a particular option. Table 10.4 shows some of the features of the benchmarks, such as the length of the paths found by reachability analysis as a function of n (the number of processes in the benchmark family). Results are reported for interleaving, step semantic and shallow synchronization.

The main finding of the experimental results is that the efficiency of the

Benchmark	Path length			Hardest instance attempted		
	Inter	Step	Shallow	Inter	Step	Shallow
Simple Ring	$5n$	6	6	5[TO]	20[1.1]	20[3.1] - 20[5.5]
Ring-shape Fischer	$7n$	7	7	5[TO]	20[8.9]	20[24.2] - 20[130.2]
Star-shape Fischer	$3n$	$3n$	$3n$	8[TO]	9[TO]	5[TO] - 6[TO]
FDDI Protocol	$2n+1$	5	3..5	15[TO]	15[TO]	20[0.7] - 20[7.3]
Nuclear Reactor	$4n$	$4n$	$4n$	8[TO]	8[TO]	6[TO] - 7[TO]
Motorcycle	$4n+3$	$4n+3$	7..9	7[TO]	6[TO]	20[22.4] - 20[259.5]
ETCS	NA	NA	17	2[TO]	2[TO]	7[TO] - 14[TO]
Multi-Frequency	NA	$3(n-1)..3n$	9	4[TO]	8[TO]	20[20.4] - 20[115.6]

Table 10.4: Columns 2, 3 and 4 report the length of the path found with the different semantics in function of the number of processes n. Columns 5, 6, 7 report the size of the hardest instance attempted, and, in square brackets, the corresponding time, or "TO" in case of timeout. For Shallow, we report the best and worst result over the different options.

bounded model checker depends on necessary depth of the search regardless the adopted semantics. The interleaving performs better than shallow synchronization in the cases where the depth of the search is the same for the different semantics (because one process interacts with all the others and its local run of one process interleaves the synchronization with all other processes): in these cases, the shallow synchronization is penalized by the overhead of the synchronizing constraints. Nevertheless, in many cases (see Fig. 10.2), the length of local runs do not depend on the number of processes. Thus, using the shallow semantics, we reach the target at same depth. In these cases, the encoding based on shallow synchronization scales exponentially better than the one based on interleaving. The shorter depth of the encoding pays off the overhead due to the more complex synchronizing constraints. The same happens for the step semantics, which is the winner when it is possible to parallelize independent transitions. Among the different options of the shallow synchronization encodings, there is no winner, but using the local encoding added after reaching the target seems to win in most of cases.

We also compared our implementation with BACH [BLW+10], which results to be faster on many examples, while on others it does not terminate with few processes. The comparison does not help in understanding which encoding is more efficient, but rather it confirms that explicit-state search is faster on automata with a small graph, while does not compete on automata with complex graph structure. Finally, we played with different search strategies but they do not modify the outcome of the presented results. All results, together with the binaries and test cases necessary to reproduce them, are available at `http://es.fbk.eu/people/tonetta/tests/forte10/`.

10.3.2 K-induction and Implicit Predicate Abstraction

Evaluation Settings

We evaluated the verification technique based on k-induction and implicit predicate abstraction presented in Section 6.3.

We evaluated the algorithm considering four different variants. All the four variants uses the MathSAT SMT solver, and the refinement based on interpolation. The first variant, K-Ind+IA, implements the basic approach, without optimization. The second variant, K-Ind+IA+Red, implements the heuristic that reduces the number of predicates. The third variant, K-Ind+IA+Inv, implements an heuristic that discovers inductive invariants: the algorithm checks, for each predicate found by interpolation, if the predicate is an inductive invariant. The last variant uses both optimization (i.e. the reduction of predicates and the heuristic invariant generation).

We compared the approach with k-induction performed in the concrete space (K-Ind) and with the implementation of IC3 [CG12] in the concrete state space. Then, we also report the result obtained using a recent approach [CGMT14a], where we integrate IC3 with implicit predicate abstraction (IC3+IA).

We performed the evaluation on two different classes of benchmarks. First, we used several hybrid systems benchmarks. In details, we used *Audio Protocol*, *FDDI Protocol*, *CSMA/CD*, *Distributed Controller*, *Star-shape Fischer*, *Fischer timed*, *Nuclear Reactor* and *Train Gate Controller.* We scaled the number of the benchmarks increasing the number of processes involved (apart for the *FDDI Protocol*). We mostly considered safe properties, but in the test cases we also had some unsafe properties. In total, we generated 180 instances.

Then, we considered several benchmarks taken from different sources: the Software Verification Competition SV-COMP [Bey13], interpreting them over Bit Vectors as done in [CGMT14a], the instances from the test suite of InvGen [GR09], the benchmarks used in [CG12] and the benchmarks of the KIND model checker [KT11]. These benchmarks contains integer, bit-vector and real type variables operations. Thus, we did not include in the evaluation IC3, which works only if the transition system is expressed using *Linear Arithmetic over Rationals.* In total, we considered 900 instances, with true and false properties. All results, together with the binaries and test cases necessary to reproduce them, are available at `http://es.fbk.eu/people/mover/tests/KINDIA/kindia.tar.bz2.`

Results

We have run our experiments on a cluster of Linux machines with a 2.27GHz Xeon CPU, using a timeout of 900 seconds and a memory limit of 4Gb for each instance.

In Figure 10.3.2 we compare the optimization of K-IND+IA for the hybrid automata benchmarks: the version of K-IND+IA that performs both the reduction and the simple invariant discovery (K-Ind+IA+Red+Inv) seems to perform better than all the other variants. In fact, it seems that for hybrid automata benchmarks the approach may find several invariants on the continuous variables. For example, in several cases the variables increase monotonically and are never set to a negative value in a discrete transition. Thus, the approach may find that a variable is always positive. These kind of simple invariants may help the inductive step of k-induction to prove that a property holds. Also, note that the reduction of the number of predicates is effective. In fact K-Ind+IA+Red+Inv solves more instances than K-Ind+IA+Inv and is usually faster.

In Figure 10.3.2 we compare K-Ind+IA+Red+Inv, the best configuration among the K-Ind+IA approaches, with the other verification algorithms, K-Ind, IC3+IA and IC3. First, K-Ind+IA+Red+Inv outperforms K-Ind on most of the benchmarks, demonstrating the effectiveness of implicit predicate abstraction. For a class of benchmarks (*FDDI Protocol*) and one of its safety properties instead K-Ind performs consistently better than K-Ind+IA+Red+Inv. In this case, K-Ind is able to prove the property without the need of any abstraction, while K-Ind+IA finds an abstraction that is harder to prove. Then, we see that the recent approaches based on IC3 are more effective than the one based on K-Ind. While this is not surprising in general, what it worth to note is that IC3 seems more efficient than IC3+IA on the hybrid automata benchmarks.

Analyzing the results obtained on the software verification benchmarks, we see from Figure 10.3.2 that the reduction of the number of predicates is crucial for K-Ind+IA to obtain good performances. Instead, the heuristic used to discovery invariant is not effective for this class of benchmarks. Comparing K-Ind+IA with K-Ind, from Figure 10.3.2, we see that K-Ind cannot solve most of the instances that K-Ind+IA+Red+Inv can solve. As before, the implicit abstraction technique based on IC3 solves more instances that K-Ind+IA+Red+Inv cannot solve before the timeout. However, on several instances where the property does not hold K-Ind+IA+Red+Inv is faster than IC3+IA.

10.4 Scenario

10.4.1 Evaluation Settings

We compared the scenario-based encodings with the different approaches based on the automata construction [CMT11a]. After the reduction of the problem to reachability using an automata encoding of the scenario, we rely on the implementation of Bounded Model Checking and K-induction of nuXmv, which uses MathSAT as the implementation of the scenario-based encoding, and is incremental (i.e. the SMT solver state is not reset when increasing the depth of the BMC search).

In the experimental evaluation, we used the following benchmarks: *Distributed Controller*, *Nuclear Reactor*, *Electronic Height Control System*, *Audio Protocol*, *Star-shape Fischer* and *Ring-shape Fischer*.

First we compare the scenario-based encoding with the automata-based approach on feasible MSCs. Then we evaluate the scenario-driven induction with the k-induction performed on the system composed with the monitor. The experimental comparison does not take into account the computation of unfeasibility explanations. On the one hand, the extraction of explanation does not appear to be straightforward for the automata-based approach (i.e. it is not clear how one can extract the same information applying the automata encoding). On the other hand, the overhead largely depends on the fact that the SMT solver must be run with proof logging activated. This can in general lead to non-negligible overheads, but in the benchmarks we analyzed this did not turn out to be the case.

The experiments were run on two Linux machine (with an Intel i7 CPU

2.93 for feasible MSCs and an Intel Core 2 Quad CPU 2.66 for unfeasible MSCs), setting the timeout and the memory out for a single run to 900 seconds and to 4 GB. The test cases, the executable and the results are available at `http://es.fbk.eu/people/mover/tests/FMSD11/`.

10.4.2 Results - Feasibility

We compare the scenario-based approach with the automata-based approach on a set of feasible meaningful scenarios, which describe the interaction of all the automata in the benchmarks, possibly containing parallel event synchronizations. We evaluate the scalability of the proposed approaches with respect to the number of components in the network and to the length of the MSCs. We increase the number of the components for all the benchmarks, except for the *Electronic Height Control System* and the *Audio Protocol*, which have a fixed number of processes.

For the automata-based approach we exploit two different construction of the monitor. In the approach called GLOBAL we construct a single monitor that represents one of the possible equivalent partial-orders imposed by the scenario, while in DISTRIBLOCAL we build a distributed monitor, one for each hybrid automaton in the network. In both construction, we used the optimization of step semantics. The details on the automata construction can be found in [CMT11a]. Then, we check the reachability of the target state of the monitor in the model obtained composing the monitor with the original system. The search is performed using an incremental BMC.

For the scenario approach, we evaluate two different variant: one is the plain scenario encoding, called SCENARIO, while the other is the variant (called SCENARIOSIMPL) where we statically compute the invariants using BDD-based reachability. Note that, in this case, the reported run times already includes the time needed to compute the invariants via BDD reachability on a discrete abstraction of the hybrid automata.

Additionally, we evaluate for both the automata-based and the scenario-based approach the optimization where we alternate the timed and the discrete transitions in the encoding. We add the suffix Alt to the names of the approaches to denote this variant.

The main findings of the experimental evaluation regard the effectiveness of the scenario-based encoding, which outperforms the optimized automata-based techniques. The Figure 10.7 (a) shows a cactus plot (in logarithmic scale) for all the tested instances of benchmarks and scenarios. The plot shows the cumulative time (on the y axes, in seconds) to solve a given number of instances (on the x axes). From the plot it is clear that the scenario-driven encoding solves more instances than the automata approaches, and is significantly faster. Moreover, we note that the alternation improves the performances for the scenario and for the global automaton, while it is counterproductive for DistribLocal.

Figure 10.7 (b) shows a scatter plot that compares the run-times of the best overall configurations available for both the automata and the scenario algorithms (DistribLocalAlt and ScenarioInvAlt). A point in the scatter plot represents the time used by DistribLocalAlt (x axes) and by ScenarioInvAlt (y axes) to solve an instance. ScenarioInvAlt outperforms DistribLocalAlt in almost all the benchmarks, solving several instances that the DistribLocalAlt cannot solve within the timeout.

Now, we evaluate the scalability of the scenario approach with respect to the number of automata in the network and with respect to the length of the scenario specifications.
In Figure 10.8 we show the plots for three benchmarks (*Star-shape Fischer*,*Ring-shape Fischer*,*Nuclear Reactor*), where on the y axes we plot the run time (in seconds) and on the x axes the number of automata in the

benchmark. Each point represents the time took by one of the approaches to solve the problem for a fixed number of processes. For each benchmark, we show the result fixing the length of the verified scenario. The plots show that the scenario based encoding scales increasing the number of automata in the network, while it is not the case for the automata based approach. Several aspects justify the increased efficiency. First, the encoding of the MSC model checking problem in the scenario-driven approach is done keeping the encoding of each single automaton local. This helps, since it allow us to keep the unrolling of the single automaton shorter (i.e. the automata in the network are unrolled for a shorter length with respect to the length of an interleaving path). This is not the case with the automata approach. Second, the search performed on the scenario-driven encoding does not have to face with the problem of finding a good interleaving of shared actions, but rather it has to find consistent local times between fixed synchronization points of the encoding. Third, the encoding simplified by the scenario results in an easier problem for the SMT solver. Comparing the optimization on the different scenario encodings, we see that on these three benchmarks both the invariant generation and the alternation of the discrete and continuous steps do not help to improve the performance. Instead, since the invariant are not effective to speed up the search, they worse the run time, due to the additional time needed to compute them (that depends on the number of automata in the network).

We show the scalability with respect to the length of the scenario in Figure 10.9 for two benchmarks, *Audio Protocol* and *Electronic Height Control System*. For the *Audio Protocol* benchmark, all the automata approaches have poor scalability, while the scenario-based encoding scales well increasing the length of the scenario. However, note how both alternation and the invariant optimization pay off in this case. For the *Electronic Height Control System* benchmark the construction based on the local monitors is effective. However, the configuration where we enable both the additional invariants and where we alternate the discrete and continuous steps, ScenarioInvAlt, outperforms all the other approaches and scale well increasing the length of the scenario.

10.4.3 Results - Unfeasibility

We compared the scenario-based induction with k-induction applied to the monolithic encoding of the network of HAs and the automata translated from the MSC. The monolithic encoding is obtained composing the network with the automata obtained from the MSC, using the DistribLocal construction with step semantics.

In order to test the scalability of both approaches, we considered a set of unfeasible MSCs of different lengths, and parameterized the number of HAs in the network. The scatter plot in Figure 10.10 shows the execution time for both methods on all the instances. The Scenario-based induction is clearly superior to monolithic k-induction. This because it exploits the structure of the MSC, resulting in localized simple path conditions, which are both simpler, and more effective, so that unsatisfiability is detected with a much shorter unrolling.

10.5 Parameter Synthesis

10.5.1 Evaluation Settings

We implemented the parameter synthesis in the nuXmv model checker. The implementation is based on the fully symbolic SMT-based IC3 of [CG12] and uses MathSAT [CGSS13] as backend SMT engine, working transition systems with linear arithmetic constraints.

Our evaluation consists of three parts. In the first, we compare our implementation (called ParamIC3 in what follows) with the approach described in [CPR08], in order to evaluate the viability of our technique when compared to other SMT-based solutions. For this, we have implemented the algorithm described in [CPR08] using our "regular" SMT-based IC3 implementation as the backend engine for reachability checking (called Iterative-Block-Path(IC3) in what follows). We remark that the tool of [CPR08] was based only on Bounded Model Checking (BMC), and exploited domain-specific information for computing the maximum needed bound, which is not available in our more general context.

In the second part, we evaluate the effectiveness of the optimization related to the computation of bad parameters, by comparing the default heuristic used by ParamIC3, using both the full counterexample path π and its initial state $(s_0, 0)$ for blocking bad regions of parameters, with the basic strategy using only $(s_0, 0)$ (called ParamIC3-basic in the following). In particular, the default heuristic used by ParamIC3 works as follows. At the beginning, only initial states $(s_0, 0)$ of counterexample paths are used to block bad regions of parameters. If the algorithm starts enumerating too many bad regions, it starts exploiting also full paths π, by computing the bad region $\beta_k^\pi(U) = \exists X.BMC_k^\pi$, where k is the length of π, and BMC_k^π is the formula encoding all the counterexample traces of length k where the values for the Boolean variables are the same as in π, similarly to what is done in [CPR08]. The computation of β_k^π is aborted if it becomes too expensive[14] , in order to control the tradeoff between the quality of the obtained bad region and the cost of performing quantifier elimination.

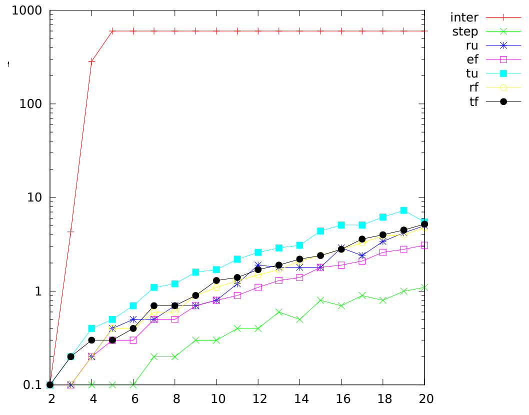
Finally, in the third part of our evaluation, we compare ParamIC3 against Red [Wan05], a state-of-the-art tool for parameter synthesis for linear-hybrid automata.

[14]We currently use a cutoff value on the number of elementary operations in the quantifier elimination module of MathSAT for this.

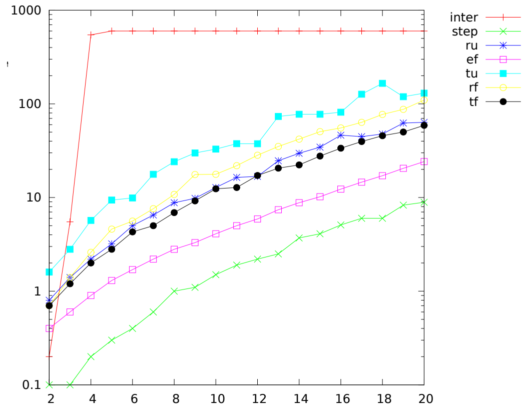
We have selected a set of the benchmark problems used in previous work on parameter synthesis for hybrid systems. Most of them come from the suite of RED. We have a total of 92 instances from 13 different families. All the instances, the scripts and the tools used for reproducing our experiments are available at `http://es.fbk.eu/people/mover/fmcad13.tar.gz`. For the first two parts of our evaluation, we have experimented with two different ways of encoding linear hybrid automata into symbolic transition systems, resulting in a set of 192 instances. For the comparison with RED, we picked the encoding giving the best overall performance for PARAMIC3.

10.5.2 Results

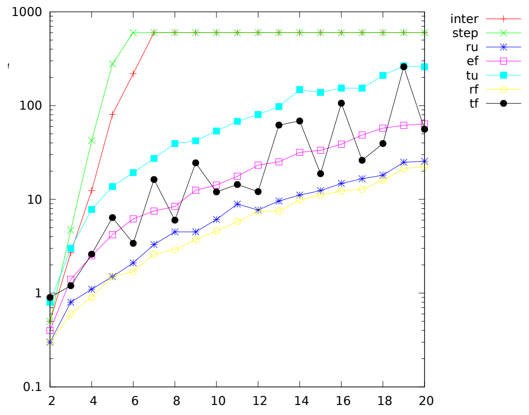
We have run our experiments on a cluster of Linux machines with a 2.27GHz Xeon CPU, using a timeout of 600 seconds and a memory limit of 3Gb for each instance. Figure 10.11 shows the scatter plots that compare the total run time (in seconds) of the different techniques. From the plots, we can make the following observations. (i) Our new algorithm is clearly superior to the technique of [CPR08], both in number of completed instances and in execution time. Overall, PARAMIC3 successfully solves 5 more instances than ITERATIVE-BLOCK-PATH(IC3), and it is almost always faster. We remark that both algorithms use the same implementation of IC3 as backend, run with the same options. (ii) Our heuristic for using full counterexample paths π for blocking bad regions of parameters pays off for harder problems. With it, PARAMIC3 solves 6 more instances which were previously out of reach, without any overhead for the other instances. (iii) The comparison with RED shows that our technique is very promising. Although there is no clear winner, there are more instances for which PARAMIC3 outperforms RED than the converse. In general, the two tools seem to be somewhat complementary. We remark that RED is specialized for timed and linear-hybrid automata and that most of the benchmarks we used come from its suite, whereas ParamIC3 works for arbitrary transition systems and it is not tuned for linear hybrid systems in any way.



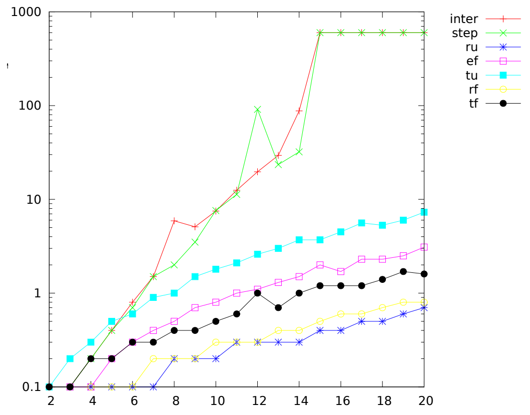
(a : Simple ring)



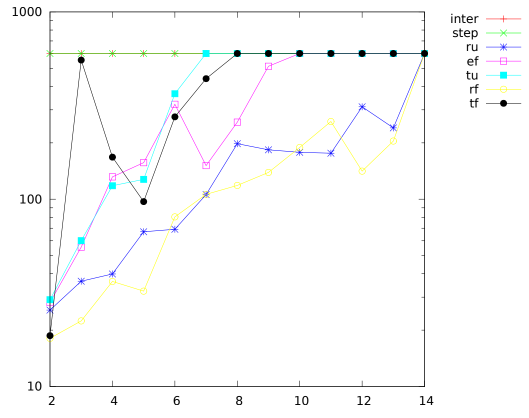
(b : Ring-shape Fischer)



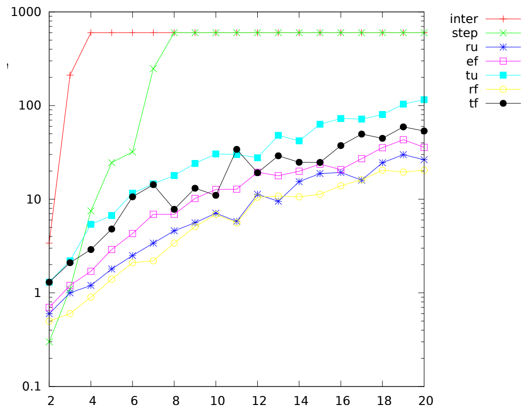
(c : Motorcycle)



(d : FDDI Protocol)



(e : ETCS)



(f : Multi frequency)

Figure 10.2: Scalable plots showing the run times (seconds on the y axes) to solve a problem with an increasing number of processes (on the x axes). For the benchmarks showed in the Figure, the length of the local runs does not depend on the number of processes.

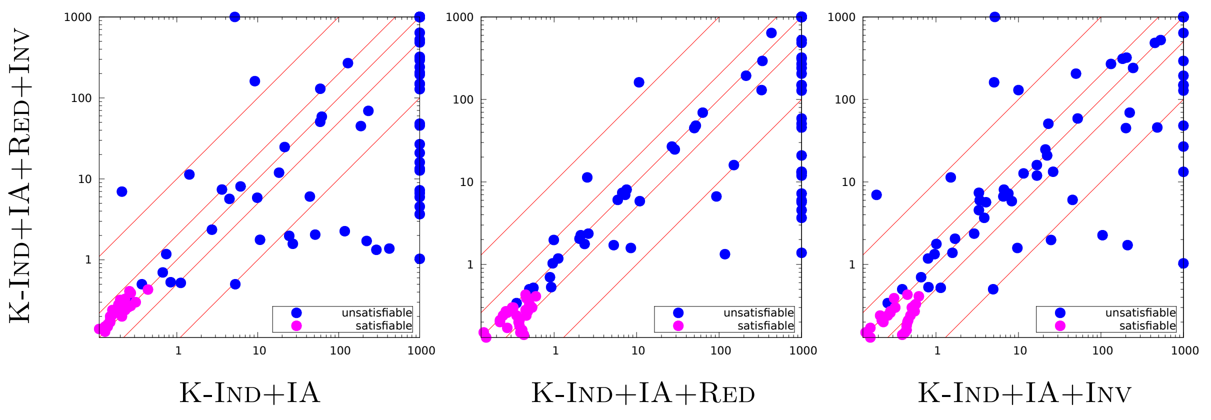


Figure 10.3: Run time comparison (sec.) between K-Ind+IA+Red+Inv and K-Ind+IA, K-Ind+IA+Red and K-Ind+IA+Inv on hybrid automata benchmarks.

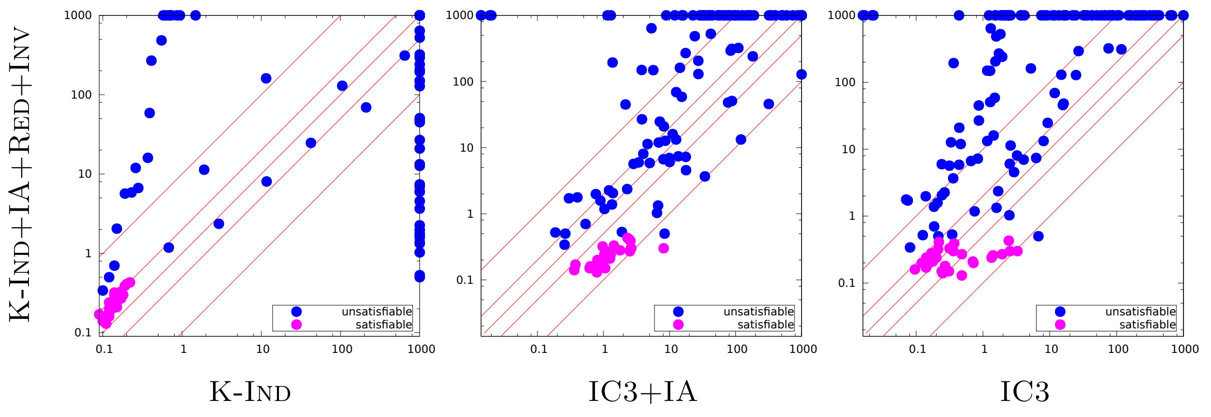


Figure 10.4: Run time comparison (sec.) between K-Ind+IA+Red+Inv and K-Ind, IC3+IA and IC3 on hybrid automata benchmarks.

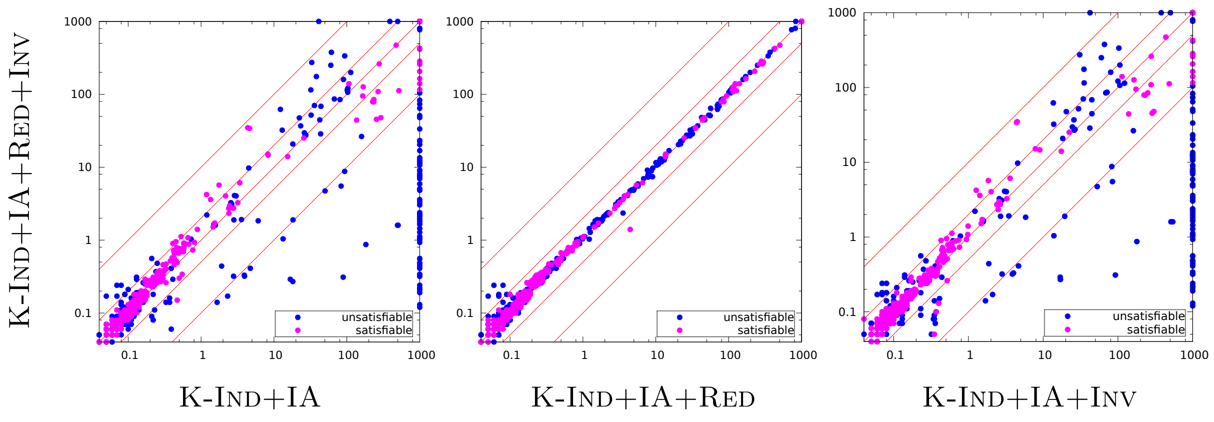


Figure 10.5: Run time comparison (sec.) between K-Ind+IA+Red+Inv and K-Ind+IA, K-Ind+IA+Red and K-Ind+IA+Inv on sw benchmarks.

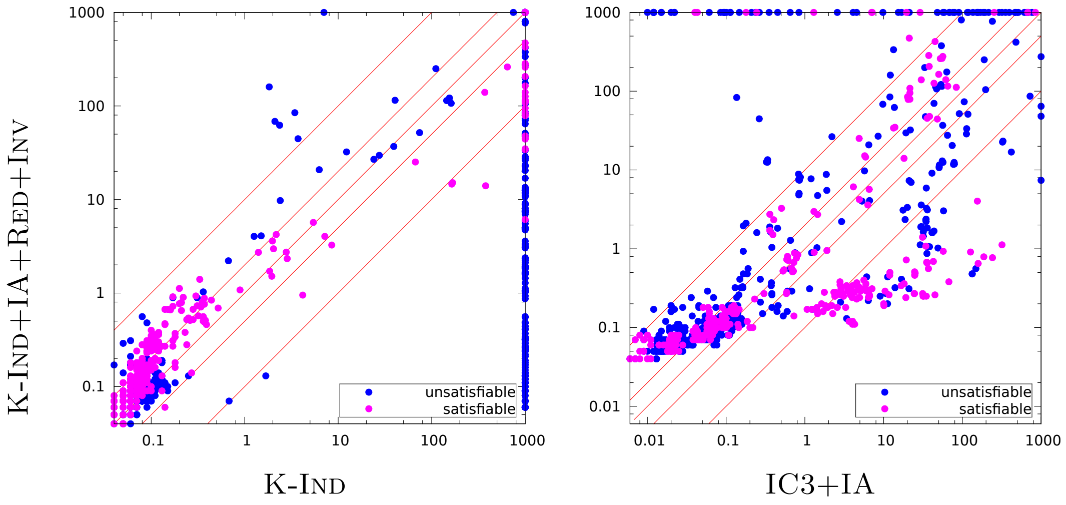
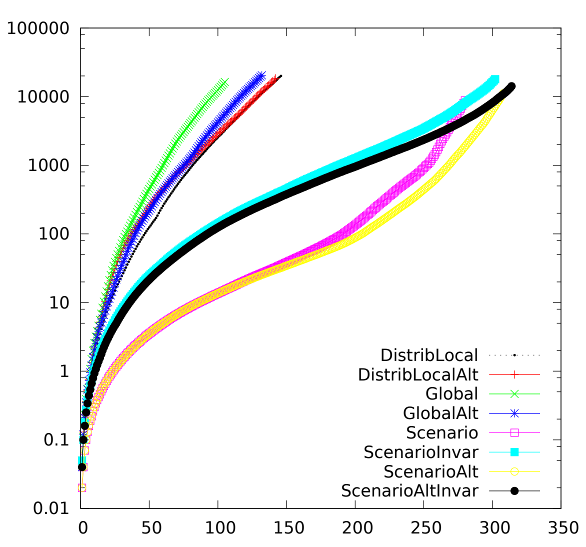
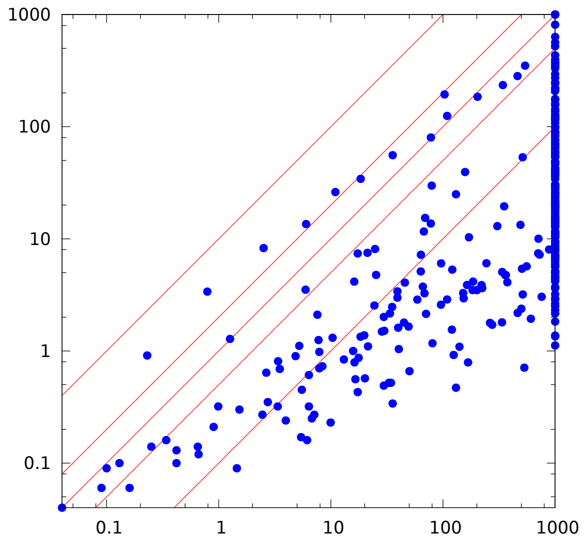


Figure 10.6: Run time comparison (sec.) between K-Ind+IA+Red+Inv and K-Ind and IC3+IA on sw benchmarks.

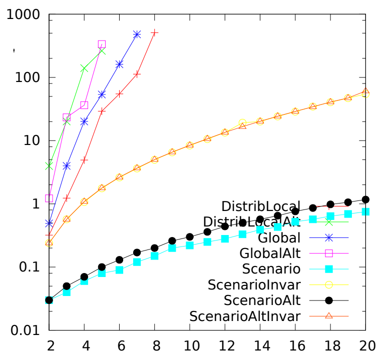


(a) Cactus plot of run times (sec.).

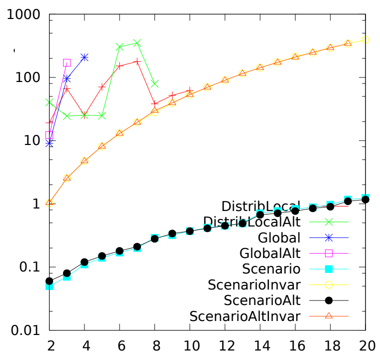


(b) Scatter plots of run times (sec.).

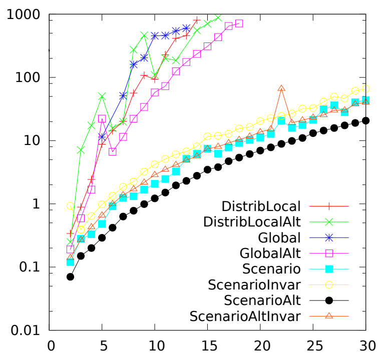
Figure 10.7: (a) Cactus plot with the cumulative time (y axes) and # of solved instances (x axes); (b) Scatter plots with ScenarioInvAlt (y axes) vs. DistribLocalAlt (x axes)



(a) *Star-shape Fischer*

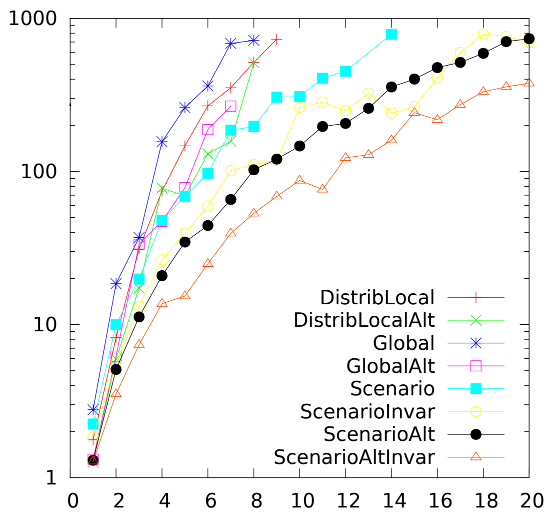


(b) *Ring-shape Fischer*

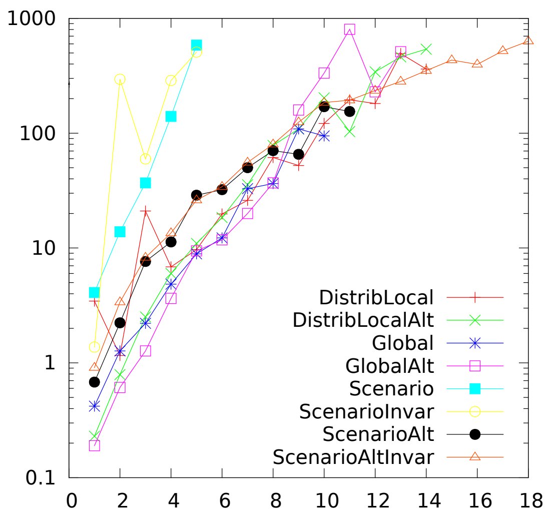


(c) *Nuclear Reactor*

Figure 10.8: Scalable plots showing the run times (sec. on the y axes) to solve a problem with an increasing the number of processes (on the x axes).



(a) *Audio Protocol*



(b) *Electronic Height Control System*

Figure 10.9: Scalable plots showing the run times (sec. on the y axes) to solve a problem with a scenario of increasing length (on the x axes).

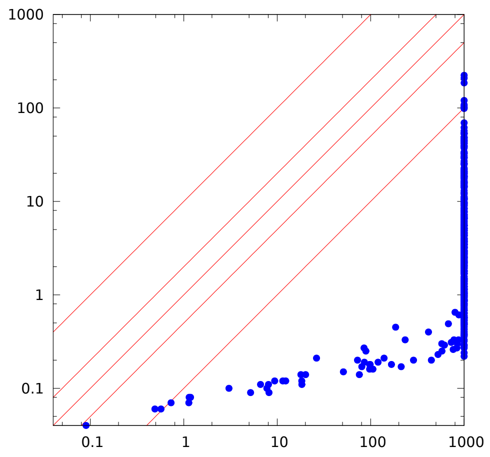


Figure 10.10: Run times (sec.): monolithic induction (x axes) vs. scenario-induction (y axes).

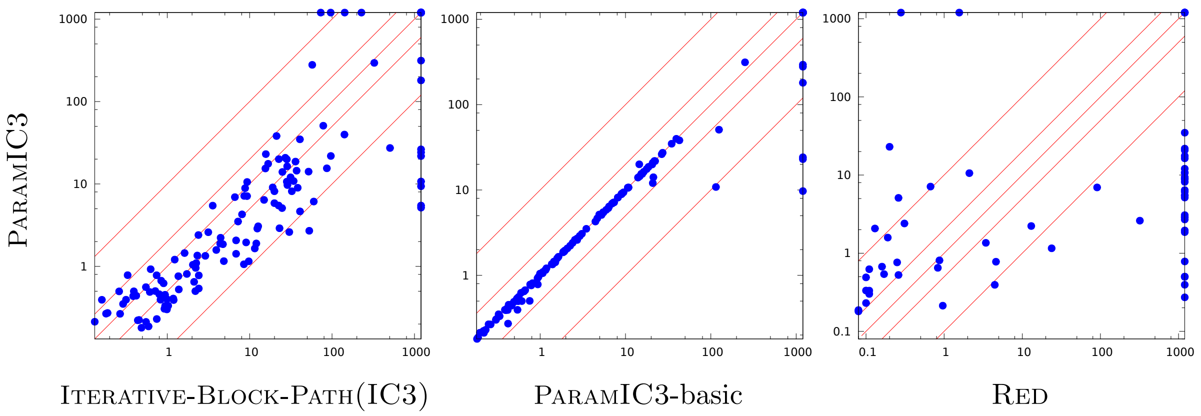


Figure 10.11: Run time comparison (sec.) between ParamIC3 and Iterative-Block-Path(IC3), ParamIC3-basic and Red.

Part V

Conclusion

Chapter 11

Conclusion and Future Work

11.1 Conclusion

A key issue during the design of an embedded systems is to identify errors. Since embedded systems are used in safety critical applications, an error may cause their failure, and thus, possibly a huge loss of money. *Formal verification* helps to find bugs in the early design phases of the system. Moreover, the adoption of formal techniques is required to adhere to certification standards (e.g. see the standard DO-178b [do192] for the avionics domain). However, the verification and the automatic analysis of complex embedded systems is challenges. These systems are usually *hybrid*, since they are formed by several components that interact with the physical environment.

In this thesis we investigated novel formal verification techniques for hybrid systems, exploiting the capabilities of SMT solvers.

To recap, the contributions of this thesis are the following.

First, we proposed a technique to encode hybrid automata with non-linear dynamics in an encoding that does not rely on quantifiers. The technique is general enough to be applied automatically to *Polynomial Hybrid Automata* and to sub-classes of *Linear Hybrid Systems*. Moreover, we also showed that the approach can be applied to some non-linear hybrid systems, although manually. The approach extends the applicability of the existing verification techniques based on SMT, such as Bounded Model Checking. While the performance of the current SMT solvers may not be satisfactory on the problems generated with this encoding, the approach is an enabler for future research.

Second, we investigated an improvement of the relational abstraction technique for *Linear Hybrid Systems*. We proposed a novel abstraction, *time-aware relational abstraction*, that has two main advantages: it is more precise and it can be refined. We demonstrated the effectiveness of the abstraction on several interesting case studies that cannot be verified with the previous approach.

Third, we proposed a Bounded Model Checking algorithm that, exploiting the structure of the hybrid automata network, may obtain performance improvements over a monolithic BMC approach.

Fourth, we showed an abstraction-refinement technique based on implicit predicate abstraction and k-induction. The refinement of the abstraction is fully incremental and enables for interesting optimization, like the reduction of predicates. With an experimental evaluation we showed that the technique is effective in proving invariant properties on infinite-state transition systems, also from the software domain.

Fifth, we proposed novel algorithms to solve the scenario-verification problem. Scenario are an important tool for the validation of a design, since they allow the user to test the feasibility of complex interactions in the system. We proposed two algorithms that exploit the hybrid automata network with the goal of increasing the performance of the verification task. We empirically showed that the algorithms work well compared to the state of the art. Moreover, we exploited the SMT framework to extract debug information in the case a scenario is unfeasible.

Sixth, we developed a novel algorithm that solve the parameter synthesis problem. Also in this case, we showed empirically that the new approach is competitive with the state of the art.

In conclusion, the thesis demonstrate three findings. First, it is possible to analyze hybrid systems with complex dynamics (beyond the linear hybrid automata case) using the SMT framework. Then, the thesis shows that the hybrid automata network formalism can be exploited to design better verification algorithms that are aware of the input problem, like the structure of the automata network. Finally, the thesis shows that the SMT framework is suitable to design efficient verification and analysis algorithms for infinite-state transition systems.

11.2 Future work

The work presented in the thesis open for several future research directions.

First, to effectively apply verification to hybrid systems in an industrial context (e.g. in the analog-mixed signals domain), there is the need to handle complex and non-linear dynamics efficiently. The approach presented in Chapter 3 gives a partial solution to the problem. However, as shown by our experimental evaluation, there is the need of more scalable verification algorithms able to reason on infinite-state transition systems expressed in the *Theory of Reals*. To improve the existing techniques, a possible solution is to develop more efficient decision procedures for the *Theory of Reals*. Another approach to solve the problem is to investigate the use of the *Theory of Reals* in the recently proposed verification algorithms, such as IC3 (e.g. note that in IC3 a single satisfiability query is less demanding compared to other approaches like, e.g. BMC). Moreover, a promising research direction regards relational abstraction (Chapter 5), which could be extended to handle non-linear hybrid systems.

Moreover, there is a demand from industrial users to solve more expressive problems than invariant verification. For example, the SMT-based techniques could be extended to verify LTL properties or HRELTL properties [CRT09].

Regarding time-aware relational abstraction, we plan to investigate several open problems. For example, the proposed technique seems amenable for an abstraction-refinement approach. However, an open question is to find an effective simulation and refinement technique. Then, we would like to apply the technique to networks of hybrid automata.

In the thesis we investigated verification techniques that exploit the structure of the automata network, with the goal of improving the scalability of the approaches. The experimental results were promising, and demonstrate that it could be interesting to push the research efforts in this direction. For example, we could investigate invariant property algorithms that exploit the structure of the automata network.

Bibliography

[ABCS05] Gilles Audemard, Marco Bozzano, Alessandro Cimatti, and Roberto Sebastiani. Verifying industrial hybrid systems with MathSAT. *Electr. Notes Theor. Comput. Sci.*, 119(2):17–32, 2005.

[ABG07] S. Akshay, Benedikt Bollig, and Paul Gastin. Automata and logics for timed message sequence charts. In Vikraman Arvind and Sanjiva Prasad, editors, *FSTTCS*, volume 4855 of *Lecture Notes in Computer Science*, pages 290–302. Springer, 2007.

[ÁBKS05] Erika Ábrahám, Bernd Becker, Felix Klaedtke, and Martin Steffen. Optimizing bounded model checking for linear hybrid systems. In Radhia Cousot, editor, *VMCAI*, volume 3385 of *Lecture Notes in Computer Science*, pages 396–412. Springer, 2005.

[ABS01] Aurore Annichini, Ahmed Bouajjani, and Mihaela Sighireanu. Trex: A tool for reachability analysis of complex systems. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *CAV*, volume 2102 of *Lecture Notes in Computer Science*, pages 368–372. Springer, 2001.

[ACD90] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking for real-time systems. In *LICS*, pages 414–425. IEEE Computer Society, 1990.

[ACHH92] Rajeev Alur, Costas Courcoubetis, Thomas. A. Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, pages 209–229, 1992.

[ACKS02] Gilles Audemard, Alessandro Cimatti, Artur Kornilowicz, and Roberto Sebastiani. Bounded model checking for timed systems. In Doron Peled and Moshe Y. Vardi, editors, *FORTE*, volume 2529 of *Lecture Notes in Computer Science*, pages 243–259. Springer, 2002.

[ADI06] Rajeev Alur, Thao Dang, and Franjo Ivancic. Counterexample-guided predicate abstraction of hybrid systems. *Theor. Comput. Sci.*, 354(2):250–271, 2006.

[ADM02] Eugene Asarin, Thao Dang, and Oded Maler. The d/dt tool for verification of hybrid systems. In Brinksma and Larsen [BL02], pages 365–370.

[ADMB00] Eugene Asarin, Thao Dang, Oded Maler, and Olivier Bournez. Approximate reachability analysis of piecewise-linear dynamical systems. In Lynch and Krogh [LK00], pages 20–31.

[AFKS12] Étienne André, Laurent Fribourg, Ulrich Kühne, and Romain Soulat. Imitator 2.5: A tool for analyzing robustness in scheduling problems. In Dimitra Giannakopoulou and Dominique Méry, editors, *FM*, volume 7436 of *Lecture Notes in Computer Science*, pages 33–36. Springer, 2012.

[AGH$^+$00] Rajeev Alur, Radu Grosu, Yerang Hur, Vijay Kumar, and Insup Lee. Modular specification of hybrid systems in charon. In Lynch and Krogh [LK00], pages 6–19.

[AK12] Étienne André and Ulrich Kühne. Parametric analysis of hybrid systems using HyMITATOR. In *iFM*, pages 16–19, 2012.

[Alu11] Rajeev Alur. Formal verification of hybrid systems. In Samarjit Chakraborty, Ahmed Jerraya, Sanjoy K. Baruah, and Sebastian Fischmeister, editors, *EMSOFT*, pages 273–278. ACM, 2011.

[AN95] Vangalur S. Alagar and Maurice Nivat, editors. *Algebraic Methodology and Software Technology, 4th International Conference, AMAST '95, Montreal, Canada, July 3-7, 1995, Proceedings*, volume 936 of *Lecture Notes in Computer Science.* Springer, 1995.

[AP04] Rajeev Alur and George J. Pappas, editors. *Hybrid Systems: Computation and Control, 7th International Workshop, HSCC 2004, Philadelphia, PA, USA, March 25-27, 2004, Proceedings*, volume 2993 of *Lecture Notes in Computer Science.* Springer, 2004.

[AY99] Rajeev Alur and Mihalis Yannakakis. Model checking of message sequence charts. In Jos C. M. Baeten and Sjouke Mauw, editors, *CONCUR*, volume 1664 of *Lecture Notes in Computer Science*, pages 114–129. Springer, 1999.

[BAL97] Hanêne Ben-Abdallah and Stefan Leue. Timing constraints in message sequence chart specifications. In Atsushi Togashi, Tadanori Mizuno, Norio Shiratori, and Teruo Higashino, editors, *FORTE*, volume 107 of *IFIP Conference Proceedings*, pages 91–106. Chapman & Hall, 1997.

[BBB07] Alberto Bemporad, Antonio Bicchi, and Giorgio C. Buttazzo, editors. *Hybrid Systems: Computation and Control, 10th In-*

ternational Workshop, HSCC 2007, Pisa, Italy, April 3-5, 2007, Proceedings, volume 4416 of *Lecture Notes in Computer Science*. Springer, 2007.

[BBC^{+}12] Luca Benvenuti, Davide Bresolin, Pieter Collins, Alberto Ferrari, Luca Geretti, and Tiziano Villa. Ariadne: Dominance checking of nonlinear hybrid automata using reachability analysis. In Alain Finkel, Jérôme Leroux, and Igor Potapov, editors, *RP*, volume 7550 of *Lecture Notes in Computer Science*, pages 79–91. Springer, 2012.

[BBC^{+}14] Luca Benvenuti, Davide Bresolin, Pieter Collins, Alberto Ferrari, Luca Geretti, and Tiziano Villa. Assumeguarantee verification of nonlinear hybrid systems withariadne. *International Journal of Robust and Nonlinear Control*, 24(4):699–724, 2014.

[BC10] Armin Biere and Koen Claessen. Hardware model checking competition, 2010.

[BCCZ99] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without bdds. In Rance Cleaveland, editor, *TACAS*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer, 1999.

[BCGR12] Roberto Bruttomesso, Alessandro Carioni, Silvio Ghilardi, and Silvio Ranise. Automated analysis of parametric timing-based mutual exclusion algorithms. In Alwyn Goodloe and Suzette Person, editors, *NASA Formal Methods*, volume 7226 of *Lecture Notes in Computer Science*, pages 279–294. Springer, 2012.

[BCL$^+$10a] Lei Bu, Alessandro Cimatti, Xuandong Li, Sergio Mover, and Stefano Tonetta. Model checking of hybrid systems using shallow synchronization. In John Hatcliff and Elena Zucca, editors, *FMOODS/FORTE*, volume 6117 of *Lecture Notes in Computer Science*, pages 155–169. Springer, 2010.

[BCL$^+$10b] Lei Bu, Alessandro Cimatti, Xuandong Li, Sergio Mover, and Stefano Tonetta. Model checking of hybrid systems using shallow synchronization (extended version). Technical report, 2010. `http://es.fbk.eu/people/tonetta/papers/forte10/`.

[BCL$^+$11] Marco Bozzano, Alessandro Cimatti, Oleg Lisagor, Cristian Mattarei, Sergio Mover, Marco Roveri, and Stefano Tonetta. Symbolic model checking and safety assessment of altarica models. *ECEASST*, 46, 2011.

[Bey13] Dirk Beyer. Second competition on software verification - (summary of sv-comp 2013). In Piterman and Smolka [PS13], pages 594–609.

[BH11] Armin Biere and Keijo Heljanko. Hardware model checking competition, 2011.

[BHSW12] Armin Biere, Keijo Heljanko, Martina Seidl, and Siert Wieringa. Hardware model checking competition, 2012.

[BHSW13] Armin Biere, Keijo Heljanko, Martina Seidl, and Siert Wieringa. Hardware model checking competition, 2013.

[BJ06] Thomas Ball and Robert B. Jones, editors. *Computer Aided Verification, 18th International Conference, CAV 2006, Seat-*

tle, WA, USA, August 17-20, 2006, Proceedings, volume 4144 of *Lecture Notes in Computer Science*. Springer, 2006.

[BJLY98] Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. Partial order reductions for timed systems. In Davide Sangiorgi and Robert de Simone, editors, *CONCUR*, volume 1466 of *Lecture Notes in Computer Science*, pages 485–500. Springer, 1998.

[BL02] Ed Brinksma and Kim Guldstrand Larsen, editors. *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, volume 2404 of *Lecture Notes in Computer Science*. Springer, 2002.

[BL11] Lei Bu and Xuandong Li. Path-oriented bounded reachability analysis of composed linear hybrid systems. *International Journal on Software Tools for Technology Transfer*, 13(4):307–317, 2011.

[BLL$^+$95] Johan Bengtsson, Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Uppaal - a tool suite for automatic verification of real-time systems. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Hybrid Systems*, volume 1066 of *Lecture Notes in Computer Science*, pages 232–243. Springer, 1995.

[BLN03] Dirk Beyer, Claus Lewerentz, and Andreas Noack. Rabbit: A tool for bdd-based verification of real-time systems. In Jr. and Somenzi [JS03], pages 122–125.

[BLR05] Gerd Behrmann, Kim Guldstrand Larsen, and Jacob Illum Rasmussen. Beyond liveness: Efficient parameter synthesis for time bounded liveness. In Paul Pettersson and Wang Yi, editors, *FORMATS*, volume 3829 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2005.

[BLW$^+$10] Lei Bu, You Li, Linzhang Wang, Xin Chen, and Xuandong Li. Bach 2 : Bounded reachability checker for compositional linear hybrid systems. In *DATE* [DBL10], pages 1512–1517.

[BM07a] Aaron R. Bradley and Zohar Manna. *The calculus of computation - decision procedures with applications to verification.* Springer, 2007.

[BM07b] Aaron R. Bradley and Zohar Manna. Checking safety by inductive generalization of counterexamples to induction. In *FMCAD* [DBL07], pages 173–180.

[BM09] Ahmed Bouajjani and Oded Maler, editors. *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, volume 5643 of *Lecture Notes in Computer Science*. Springer, 2009.

[BPR03] Thomas Ball, Andreas Podelski, and Sriram K. Rajamani. Boolean and cartesian abstraction for model checking c programs. *STTT*, 5(1):49–58, 2003.

[BPST10] Roberto Bruttomesso, Edgar Pek, Natasha Sharygina, and Aliaksei Tsitovich. The opensmt solver. In Esparza and Majumdar [EM10], pages 150–153.

[Bra11] Aaron R. Bradley. Sat-based model checking without unrolling. In Ranjit Jhala and David A. Schmidt, editors, *VMCAI*, volume 6538 of *Lecture Notes in Computer Science*, pages 70–87. Springer, 2011.

[Bro03] Christopher W. Brown. QEPCAD B: A program for computing with semi-algebraic sets using CADs. *SIGSAM BULLETIN*, 37:97–108, 2003.

[BS10] Kerstin Bauer and Klaus Schneider. From synchronous programs to symbolic representations of hybrid systems. In Johansson and Yi [JY10], pages 41–50.

[BS11] Per Bjesse and Anna Slobodová, editors. *International Conference on Formal Methods in Computer-Aided Design, FMCAD '11, Austin, TX, USA, October 30 - November 02, 2011*. FMCAD Inc., 2011.

[BSST09] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. IOS Press, 2009.

[BZL10] Lei Bu, Jianhua Zhao, and Xuandong Li. Path-oriented reachability verification of a class of nonlinear hybrid automata using convex programming. In Gilles Barthe and Manuel V. Hermenegildo, editors, *VMCAI*, volume 5944 of *Lecture Notes in Computer Science*, pages 78–94. Springer, 2010.

[CCF$^+$07a] Alberto Casagrande, Kevin Casey, Rachele Falchi, Carla Piazza, Benedetto Ruperti, Giannina Vizzotto, and Bud Mishra. Translating time-course gene expression profiles into semi-algebraic hybrid automata via dimensionality reduction. In Hirokazu Anai, Katsuhisa Horimoto, and Temur Kutsia, editors, *AB*, volume 4545 of *Lecture Notes in Computer Science*, pages 51–65. Springer, 2007.

[CCF$^+$07b] Roberto Cavada, Alessandro Cimatti, Anders Franzén, Krishnamani Kalyanasundaram, Marco Roveri, and R. K. Shyamasundar. Computing predicate abstractions by integrating bdds and smt solvers. In *FMCAD* [DBL07], pages 69–76.

[CCG$^+$02] Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In Brinksma and Larsen [BL02], pages 359–364.

[CD09] Ana Cavalcanti and Dennis Dams, editors. *FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings*, volume 5850 of *Lecture Notes in Computer Science*. Springer, 2009.

[CDJR09] Alessandro Cimatti, Jori Dubrovin, Tommi A. Junttila, and Marco Roveri. Structure-aware computation of predicate abstraction. In *FMCAD*, pages 9–16. IEEE, 2009.

[CFG$^+$10] Alessandro Cimatti, Anders Franzén, Alberto Griggio, Krishnamani Kalyanasundaram, and Marco Roveri. Tighter integration of bdds and smt for predicate abstraction. In *DATE* [DBL10], pages 1707–1712.

[CFH$^+$03] Edmund M. Clarke, Ansgar Fehnker, Zhi Han, Bruce H. Krogh, Joël Ouaknine, Olaf Stursberg, and Michael Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *Int. J. Found. Comput. Sci.*, 14(4):583–604, 2003.

[CG12] Alessandro Cimatti and Alberto Griggio. Software model checking via ic3. In Madhusudan and Seshia [MS12], pages 277–293.

[CGJ$^+$00] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In E. Allen Emerson and A. Prasad Sistla, editors, *CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000.

[CGL94] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model Checking and Abstraction. *ACM Trans. Program. Lang. Syst.*, 16(5):1512–1542, 1994.

[CGMT13] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. Parameter synthesis with ic3. In *FMCAD* [DBL13], pages 165–168.

[CGMT14a] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. Ic3 modulo theories via implicit predicate abstraction. In *TACAS*, 2014.

[CGMT14b] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. Verifying ltl properties of hybrid systems with k-liveness. Technical report, 2014.

[CGS10] Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani. Efficient generation of craig interpolants in satisfiability modulo theories. *ACM Trans. Comput. Log.*, 12(1):7, 2010.

[CGS11] Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani. Computing small unsatisfiable cores in satisfiability modulo theories. *J. Artif. Intell. Res. (JAIR)*, 40:701–728, 2011.

[CGSS13] Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, and Roberto Sebastiani. The mathsat5 smt solver. In Piterman and Smolka [PS13], pages 93–107.

[CM06] Prakash Chandrasekaran and Madhavan Mukund. Matching scenarios with timing constraints. In Eugene Asarin and Patricia Bouyer, editors, *FORMATS*, volume 4202 of *Lecture Notes in Computer Science*, pages 98–112. Springer, 2006.

[CMT11a] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. Efficient scenario verification for hybrid automata. In Gopalakrishnan and Qadeer [GQ11], pages 317–332.

[CMT11b] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. Hydi: A language for symbolic hybrid systems with discrete interaction. In *EUROMICRO-SEAA*, pages 275–278. IEEE, 2011.

[CMT11c] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. Proving and explaining the unfeasibility of message sequence charts for hybrid systems. In Bjesse and Slobodová [BS11], pages 54–62.

[CMT12] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. A quantifier-free smt encoding of non-linear hybrid automata. In Cabodi and Singh [CS12a], pages 187–195.

[CMT13a] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. HyDI - Language Tutorial. Technical report, 2013. `https://es.fbk.eu/tools/hycomp/documents/language.pdf`.

[CMT13b] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. Quantifier-free encoding of invariants for hybrid systems. *Formal Methods in System Design*, pages 1–24, 2013.

[CMT13c] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. Smt-based scenario verification for hybrid systems. *Formal Methods in System Design*, 42(1):46–66, 2013.

[Col75] George E. Collins. Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In H. Barkhage, editor, *Automata Theory and Formal Languages*, volume 33 of *Lecture Notes in Computer Science*, pages 134–183. Springer, 1975.

[CPR08] Alessandro Cimatti, Luigi Palopoli, and Yusi Ramadian. Symbolic computation of schedulability regions using parametric timed automata. In *RTSS*, pages 80–89. IEEE Computer Society, 2008.

[CRT09] Alessandro Cimatti, Marco Roveri, and Stefano Tonetta. Requirements validation for hybrid systems. In Bouajjani and Maler [BM09], pages 188–203.

[CS12a] Gianpiero Cabodi and Satnam Singh, editors. *Formal Methods in Computer-Aided Design, FMCAD 2012, Cambridge, UK, October 22-25, 2012*. IEEE, 2012.

[CS12b] Koen Claessen and Niklas Sörensson. A liveness checking algorithm that counts. In Cabodi and Singh [CS12a], pages 52–59.

[dAM95] Luca de Alfaro and Zohar Manna. Verification in continuous time by discrete reasoning. In Alagar and Nivat [AN95], pages 292–306.

[DBL07] *Formal Methods in Computer-Aided Design, 7th International Conference, FMCAD 2007, Austin, Texas, USA, November 11-14, 2007, Proceedings.* IEEE Computer Society, 2007.

[DBL10] *Design, Automation and Test in Europe, DATE 2010, Dresden, Germany, March 8-12, 2010.* IEEE, 2010.

[DBL13] *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013.* IEEE, 2013.

[DDD$^+$12] Werner Damm, Henning Dierks, Stefan Disch, Willem Hagemann, Florian Pigorsch, Christoph Scholl, Uwe Waldmann, and Boris Wirtz. Exact and fully symbolic verification of linear hybrid automata with large discrete state spaces. *Sci. Comput. Program.*, 77(10-11):1122–1150, 2012.

[DdM06] Bruno Dutertre and Leonardo Mendonça de Moura. A fast linear-arithmetic solver for dpll(t). In Ball and Jones [BJ06], pages 81–94.

[DE73] George B. Dantzig and B. Curtis Eaves. Fourier-motzkin elimination and its dual. *Journal of Combinatorial Theory(A)*, 1973.

[DH01] Werner Damm and Dave Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[DJH12] Jori Dubrovin, Tommi A. Junttila, and Keijo Heljanko. Exploiting step semantics for efficient bounded model checking of asynchronous systems. *Sci. Comput. Program.*, 77(10-11):1095–1121, 2012.

[DM12] Parasara Sridhar Duggirala and Sayan Mitra. Lyapunov abstractions for inevitability of hybrid systems. In Thao Dang and Ian M. Mitchell, editors, *HSCC*, pages 115–124. ACM, 2012.

[dMB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[dMOR$^+$04] Leonardo Mendonça de Moura, Sam Owre, Harald Rueß, John M. Rushby, Natarajan Shankar, Maria Sorea, and Ashish Tiwari. Sal 2. In Rajeev Alur and Doron Peled, editors, *CAV*, volume 3114 of *Lecture Notes in Computer Science*, pages 496–500. Springer, 2004.

[dMRS02] Leonardo Mendonça de Moura, Harald Rueß, and Maria Sorea. Lazy theorem proving for bounded model checking over infinite domains. In Andrei Voronkov, editor, *CADE*, volume 2392 of *Lecture Notes in Computer Science*, pages 438–455. Springer, 2002.

[dMRS03] Leonardo Mendonça de Moura, Harald Rueß, and Maria Sorea. Bounded model checking and induction: From refutation to verification (extended abstract, category a). In Jr. and Somenzi [JS03], pages 14–26.

[do192] Software Considerations in Airborne Systems and Equipment Certification DO-178-B/ED-12-B. Requirements and Technical Concepts for Aviation/European Organization for Civil Aviation Equipement, 1992.

[DS96] Andreas Dolzmann and Thomas Sturm. REDLOG Computer Algebra Meets Computer Logic. *ACM SIGSAM Bulletin*, 31:2–9, 1996.

[DSW98] Andreas Dolzmann, Thomas Sturm, and Volker Weispfenning. Real quantifier elimination in practice. In *Alg. Algebra and Number Theory*, pages 221–247. Springer, 1998.

[EM10] Javier Esparza and Rupak Majumdar, editors. *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6015 of *Lecture Notes in Computer Science*. Springer, 2010.

[EMB11] Niklas Eén, Alan Mishchenko, and Robert K. Brayton. Efficient implementation of property directed reachability. In Bjesse and Slobodová [BS11], pages 125–134.

[ERNF11] Andreas Eggers, Nacim Ramdani, Nedialko Nedialkov, and Martin Fränzle. Improving sat modulo ode for hybrid systems analysis by combining different enclosure methods. In Gilles Barthe, Alberto Pardo, and Gerardo Schneider, editors, *SEFM*, volume 7041 of *Lecture Notes in Computer Science*, pages 172–187. Springer, 2011.

[ES03] Niklas Eén and Niklas Sörensson. Temporal induction by incremental sat solving. *Electr. Notes Theor. Comput. Sci.*, 89(4):543–560, 2003.

[FB02] Ian Fialho and Gary J. Balas. Road adaptive active suspension design using linear parameter-varying scheduling. *IEEE Trans. on Control Sys. Tech.*, 10(1), 2002.

[FGD$^+$11a] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. Spaceex: Scalable verification of hybrid systems. In Gopalakrishnan and Qadeer [GQ11], pages 379–395.

[FGD$^+$11b] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. Spaceex: Scalable verification of hybrid systems. In Gopalakrishnan and Qadeer [GQ11], pages 379–395.

[FH05] Martin Fränzle and Christian Herde. Efficient proof engines for bounded model checking of hybrid systems. *Electr. Notes Theor. Comput. Sci.*, 133:119–137, 2005.

[FH07] Martin Fränzle and Christian Herde. Hysat: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30(3):179–198, 2007.

[FHSW07] Martin Fränzle, Hardi Hungar, Christian Schmitt, and Boris Wirtz. Hlang: Compositional representation of hybrid systems via predicates. Reports of SFB/TR 14 AVACS 20, July 2007. ISSN: 1860-9821, http://www.avacs.org.

[FJK08] Goran Frehse, Sumit Kumar Jha, and Bruce H. Krogh. A counterexample-guided approach to parameter synthesis for linear hybrid automata. In Magnus Egerstedt and Bud

Mishra, editors, *HSCC*, volume 4981 of *Lecture Notes in Computer Science*, pages 187–200. Springer, 2008.

[Frä01] Martin Fränzle. What will be eventually true of polynomial hybrid automata? In Naoki Kobayashi and Benjamin C. Pierce, editors, *TACS*, volume 2215 of *Lecture Notes in Computer Science*, pages 340–359. Springer, 2001.

[Fre08] Goran Frehse. Phaver: algorithmic verification of hybrid systems past hytech. *STTT*, 10(3):263–279, 2008.

[GKC13] Sicun Gao, Soonho Kong, and Edmund M. Clarke. Satisfiability modulo odes. In *FMCAD* [DBL13], pages 105–112.

[God96] Patrice Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems - An Approach to the State-Explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer, 1996.

[GQ11] Ganesh Gopalakrishnan and Shaz Qadeer, editors. *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*. Springer, 2011.

[GR09] Ashutosh Gupta and Andrey Rybalchenko. Invgen: An efficient invariant generator. In Bouajjani and Maler [BM09], pages 634–640.

[GS97] Susanne Graf and Hassen Saïdi. Construction of abstract state graphs with pvs. In Orna Grumberg, editor, *CAV*, volume 1254 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 1997.

[GSV13] Luigi Di Guglielmo, Sanjit A. Seshia, and Tiziano Villa. Synthesis of implementable control strategies for lazy linear hybrid automata. In Maria Ganzha, Leszek A. Maciaszek, and Marcin Paprzycki, editors, *FedCSIS*, pages 1369–1376, 2013.

[HB12] Krystof Hoder and Nikolaj Bjørner. Generalized property directed reachability. In Alessandro Cimatti and Roberto Sebastiani, editors, *SAT*, volume 7317 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2012.

[HEFT08] Christian Herde, Andreas Eggers, Martin Fränzle, and Tino Teige. Analysis of hybrid systems using hysat. In *ICONS*, pages 196–201. IEEE Computer Society, 2008.

[Hen96] Thomas A. Henzinger. The theory of hybrid automata. In *LICS*, pages 278–292. IEEE Computer Society, 1996.

[Hen00] Thomas A. Henzinger. Masaccio: A formal model for embedded components. In Jan van Leeuwen, Osamu Watanabe, Masami Hagiya, Peter D. Mosses, and Takayasu Ito, editors, *IFIP TCS*, volume 1872 of *Lecture Notes in Computer Science*, pages 549–563. Springer, 2000.

[HH94] Thomas A. Henzinger and Pei-Hsin Ho. Hytech: The cornell hybrid technology tool. In Panos J. Antsaklis, Wolf Kohn, Anil Nerode, and Shankar Sastry, editors, *Hybrid Systems*, volume 999 of *Lecture Notes in Computer Science*, pages 265–293. Springer, 1994.

[HHWT97] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Hytech: A model checker for hybrid systems. *STTT*, 1(1-2):110–122, 1997.

[HHWT98] Thomas A. Henzinger, Pei-Hsin Ho, and H. Wong-Toi. Algorithmic Analysis of Nonlinear Hybrid Systems. 1998.

[Hil00] Roman Hilscher. Surprises about some elementary functions: uniform linear approximations. Technical report, 2000.

[HJMM04] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth L. McMillan. Abstractions from proofs. In Neil D. Jones and Xavier Leroy, editors, *POPL*, pages 232–244. ACM, 2004.

[HKPV98] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *J. Comput. Syst. Sci.*, 57(1):94–124, 1998.

[HN03] Keijo. Heljanko and Ilkka Niemelä. Bounded LTL model checking with stable models. *Theory and Practice of Logic Progr.*, 3(4-5):519–550, 2003.

[hyc] HyCOMP tool. `https://es.fbk.eu/tools/hycomp`.

[IT96] ITU-T. Recommendation Z.120 - Message Sequence Charts. 1996.

[IUH11] Daisuke Ishii, Kazunori Ueda, and Hiroshi Hosobe. An interval-based sat modulo ode solver for model checking nonlinear hybrid systems. *STTT*, 13(5):449–461, 2011.

[JBS07] Susmit Jha, Bryan A. Brady, and Sanjit A. Seshia. Symbolic reachability analysis of lazy linear hybrid automata. In Jean-François Raskin and P. S. Thiagarajan, editors, *FORMATS*, volume 4763 of *Lecture Notes in Computer Science*, pages 241–256. Springer, 2007.

[JdM12] Dejan Jovanovic and Leonardo Mendonça de Moura. Solving non-linear arithmetic. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *IJCAR*, volume 7364 of *Lecture Notes in Computer Science*, pages 339–354. Springer, 2012.

[JKWC07] Sumit Kumar Jha, Bruce H. Krogh, James E. Weimer, and Edmund M. Clarke. Reachability for linear hybrid automata using iterative relaxation abstraction. In Bemporad et al. [BBB07], pages 287–300.

[JS03] Warren A. Hunt Jr. and Fabio Somenzi, editors. *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*, volume 2725 of *Lecture Notes in Computer Science*. Springer, 2003.

[JY10] Karl Henrik Johansson and Wang Yi, editors. *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2010, Stockholm, Sweden, April 12-15, 2010*. ACM, 2010.

[KJN12a] Roland Kindermann, Tommi A. Junttila, and Ilkka Niemelä. Beyond lassos: Complete smt-based bounded model checking for timed automata. In Holger Giese and Grigore Rosu, editors, *FMOODS/FORTE*, volume 7273 of *Lecture Notes in Computer Science*, pages 84–100. Springer, 2012.

[KJN12b] Roland Kindermann, Tommi A. Junttila, and Ilkka Niemelä. Smt-based induction methods for timed systems. In Marcin Jurdzinski and Dejan Nickovic, editors, *FORMATS*, volume 7595 of *Lecture Notes in Computer Science*, pages 171–187. Springer, 2012.

[KJN13] Roland Kindermann, Tommi A. Junttila, and Ilkka Niemelä. Bounded model checking of an mitl fragment for timed automata. In *ACSD*, pages 216–225. IEEE, 2013.

[KMNP13] Johannes Kloos, Rupak Majumdar, Filip Niksic, and Ruzica Piskac. Incremental, inductive coverability. In Natasha Sharygina and Helmut Veith, editors, *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 158–173. Springer, 2013.

[KT11] Temesghen Kahsai and Cesare Tinelli. Pkind: A parallel k-induction based model checker. In Jiri Barnat and Keijo Heljanko, editors, *PDMC*, volume 72 of *EPTCS*, pages 55–62, 2011.

[KW01] Jochen Klose and Hartmut Wittke. An automata based interpretation of live sequence charts. In Tiziana Margaria and Wang Yi, editors, *TACAS*, volume 2031 of *Lecture Notes in Computer Science*, pages 512–527. Springer, 2001.

[LBD$^+$10] Shuhao Li, Sandie Balaguer, Alexandre David, Kim G. Larsen, Brian Nielsen, and Saulius Pusinskas. Scenario-based verification of real-time systems using uppaal. *Formal Methods in System Design*, 37(2-3):200–264, 2010.

[LK00] Nancy A. Lynch and Bruce H. Krogh, editors. *Hybrid Systems: Computation and Control, Third International Workshop, HSCC 2000, Pittsburgh, PA, USA, March 23-25, 2000, Proceedings*, volume 1790 of *Lecture Notes in Computer Science*. Springer, 2000.

[LL92] Peter B. Ladkin and Stefan Leue. On the semantics of message sequence charts. In Hartmut König, editor, *FBT*, pages 88–104. K. G. Saur Verlag, 1992.

[LL95] Peter B. Ladkin and Stefan Leue. Interpreting Message Flow Graphs. *Formal Asp. Comput.*, 7(5):473–509, 1995.

[LNO06] Shuvendu K. Lahiri, Robert Nieuwenhuis, and Albert Oliveras. Smt techniques for fast predicate abstraction. In Ball and Jones [BJ06], pages 424–437.

[LPY01] Gerardo Lafferriere, George J. Pappas, and Sergio Yovine. Symbolic Reachability Computation for Families of Linear Vector Fields. *J. Symb. Comput.*, 32(3):231–253, 2001.

[LSV03] Nancy Lynch, Roberto Segala, and Frits Vaandrager. Hybrid i/o automata. *Information and Computation*, 185(1):105 – 157, 2003.

[LW93] Rüdiger Loos and Volker Weispfenning. Applying linear quantifier elimination. *Comput. J.*, 36(5):450–462, 1993.

[McM93] Kenneth L. McMillan. *Symbolic model checking.* Kluwer, 1993.

[McM03] Kenneth L. McMillan. Interpolation and sat-based model checking. In Jr. and Somenzi [JS03], pages 1–13.

[McM05] Kenneth L. McMillan. Applications of craig interpolants in model checking. In Nicolas Halbwachs and Lenore D. Zuck, editors, *TACAS*, volume 3440 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005.

[MCTT13] Sergio Mover, Alessandro Cimatti, Ashish Tiwari, and Stefano Tonetta. Time-aware relational abstractions for hybrid systems. In *EMSOFT*, pages 1–10. IEEE, 2013.

[MIS] The MISSA Project. `http://www.missa-fp7.eu`.

[MN10] Janusz Malinowski and Peter Niebert. Sat based bounded model checking with partial order semantics for timed automata. In Esparza and Majumdar [EM10], pages 405–419.

[Mon10] David Monniaux. Quantifier elimination by lazy model enumeration. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *CAV*, volume 6174 of *Lecture Notes in Computer Science*, pages 585–599. Springer, 2010.

[MR97] Sjouke Mauw and Michel A. Reniers. High-level message sequence charts. In Ana R. Cavalli and Amardeo Sarma, editors, *SDL Forum*, pages 291–306. Elsevier, 1997.

[MS00] Olaf Müller and Thomas Stauner. Modelling and verification using Linear Hybrid Automata - a case study. *Mathematical and Computer Modelling of Dynamical Systems*, 71:71–89, 2000.

[MS12] Parthasarathy Madhusudan and Sanjit A. Seshia, editors. *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, volume 7358 of *Lecture Notes in Computer Science*. Springer, 2012.

[NMA$^+$02] Peter Niebert, Moez Mahfoudh, Eugene Asarin, Marius Bozga, Oded Maler, and Navendu Jain. Verification of timed automata via satisfiability checking. In Werner Damm and Ernst-Rüdiger Olderog, editors, *FTRTFT*, volume 2469 of *Lecture Notes in Computer Science*, pages 225–244. Springer, 2002.

[NSF$^+$10] Truong Nghiem, Sriram Sankaranarayanan, Georgios E. Fainekos, Franjo Ivancic, Aarti Gupta, and George J. Pappas.

Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In Johansson and Yi [JY10], pages 211–220.

[nux] nuXmv tool. `https://es.fbk.eu/tools/nuxmv`.

[PBL09] Minxue Pan, Lei Bu, and Xuandong Li. Tass: Timing analyzer of scenario-based specifications. In Bouajjani and Maler [BM09], pages 689–695.

[PC07] André Platzer and Edmund M. Clarke. The image computation problem in hybrid systems model checking. In Bemporad et al. [BBB07], pages 473–486.

[PC09] André Platzer and Edmund M. Clarke. Formal verification of curved flight collision avoidance maneuvers: A case study. In Cavalcanti and Dams [CD09], pages 547–562.

[Pik05] Lee Pike. Real-Time System Verification by k-Induction. Technical Report NASA/TM-2005-213751, NASA, 2005.

[Pik07] Lee Pike. Modeling time-triggered protocols and verifying their real-time schedules. In *FMCAD* [DBL07], pages 231–238.

[PJ04] Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. In Alur and Pappas [AP04], pages 477–492.

[PKV13] Erion Plaku, Lydia E. Kavraki, and Moshe Y. Vardi. Falsification of ltl safety properties in hybrid systems. *STTT*, 15(4):305–320, 2013.

[Pla08] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reasoning*, 41(2):143–189, 2008.

[Pla10] André Platzer. Differential-algebraic dynamic logic for differential-algebraic programs. *J. Log. Comput.*, 20(1):309–352, 2010. Advance Access published on November 18, 2008.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE Computer Society, 1977.

[PQ08] André Platzer and Jan-David Quesel. Keymaera: A hybrid theorem prover for hybrid systems (system description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR*, volume 5195 of *Lecture Notes in Computer Science*, pages 171–178. Springer, 2008.

[PS13] Nir Piterman and Scott A. Smolka, editors. *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7795 of *Lecture Notes in Computer Science.* Springer, 2013.

[Rab98] Alexander M. Rabinovich. On the decidability of continuous time specification formalisms. *J. Log. Comput.*, 8(5):669–678, 1998.

[RS07] Stefan Ratschan and Zhikun She. Safety verification of hybrid systems by constraint propagation-based abstraction refinement. *ACM Trans. Embedded Comput. Syst.*, 6(1), 2007.

[Sch09a] Klaus Schneider. The synchronous programming language quartz. Technical report, Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany, 2009. Internal Report 375.

[Sch09b] Viktor Schuppan. Towards a notion of unsatisfiable cores for ltl. In Farhad Arbab and Marjan Sirjani, editors, *FSEN*, volume 5961 of *Lecture Notes in Computer Science*, pages 129–145. Springer, 2009.

[Sch12] Viktor Schuppan. Towards a notion of unsatisfiable and unrealizable cores for ltl. *Sci. Comput. Program.*, 77(7-8):908–939, 2012.

[SD10] Wilfried Steiner and Bruno Dutertre. Smt-based formal verification of a *ttethernet* synchronization function. In Stefan Kowalewski and Marco Roveri, editors, *FMICS*, volume 6371 of *Lecture Notes in Computer Science*, pages 148–163. Springer, 2010.

[SDS08] Maria Sorea, Bruno Dutertre, and Wilfried Steiner. Modeling and verification of time-triggered communication protocols. In *ISORC*, pages 422–428. IEEE Computer Society, 2008.

[Sor02] Maria Sorea. Bounded model checking for timed automata. *Electr. Notes Theor. Comput. Sci.*, 68(5):116–134, 2002.

[SSS00] Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. Checking safety properties using induction and a sat-solver. In Warren A. Hunt Jr. and Steven D. Johnson, editors, *FMCAD*, volume 1954 of *Lecture Notes in Computer Science*, pages 108–125. Springer, 2000.

[ST11a] Sriram Sankaranarayanan and Ashish Tiwari. Relational abstractions for continuous and hybrid systems. In Gopalakrishnan and Qadeer [GQ11], pages 686–702.

[ST11b] Thomas Sturm and Ashish Tiwari. Verification and synthesis using real quantifier elimination. In Éric Schost and Ioannis Z. Emiris, editors, *ISSAC*, pages 329–336. ACM, 2011.

[STT09] Natasha Sharygina, Stefano Tonetta, and Aliaksei Tsitovich. The synergy of precise and fast abstractions for program verification. In Sung Y. Shin and Sascha Ossowski, editors, *SAC*, pages 566–573. ACM, 2009.

[SV07] Marko Samer and Helmut Veith. On the notion of vacuous truth. In Nachum Dershowitz and Andrei Voronkov, editors, *LPAR*, volume 4790 of *Lecture Notes in Computer Science*, pages 2–14. Springer, 2007.

[TD12] Ashish Tiwari and Bruno Dutertre. Modeling and analysis of asynchronous systems using SAL and Hybrid SAL. Technical report, 2012.

[tea02] MoBIES team. Hsif semantics. Technical report, University of Pennsylvania, 2002.

[Tiw08] Ashish Tiwari. Abstractions for hybrid systems. *Formal Methods in System Design*, 32(1):57–83, 2008.

[Tiw12] Ashish Tiwari. HybridSAL Relational Abstracter. In Madhusudan and Seshia [MS12], pages 725–731.

[TK04] Ashish Tiwari and Gaurav Khanna. Nonlinear systems: Approximating reach sets. In Alur and Pappas [AP04], pages 600–614.

[Ton09] Stefano Tonetta. Abstract model checking without computing the abstraction. In Cavalcanti and Dams [CD09], pages 89–105.

[Var95] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In Faron Moller and Graham M. Birtwistle, editors, *Banff Higher Order Workshop*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer, 1995.

[vBRSR07] Dirk A. van Beek, Michel A. Reniers, Ramon R. H. Schiffelers, and J. E. Rooda. Foundations of a compositional interchange format for hybrid systems. In Bemporad et al. [BBB07], pages 587–600.

[Wan05] Farn Wang. Symbolic parametric safety analysis of linear hybrid systems with bdd-like data-structures. *IEEE Trans. Software Eng.*, 31(1):38–51, 2005.

[WK13] Tobias Welp and Andreas Kuehlmann. Qf bv model checking with property directed reachability. In Enrico Macii, editor, *DATE*, pages 791–796. EDA Consortium San Jose, CA, USA / ACM DL, 2013.

[ZLZZ03] Jianhua Zhao, Xuandong Li, Tao Zheng, and Guoliang Zheng. Removing irrelevant atomic formulas for checking timed automata efficiently. In Kim Guldstrand Larsen and Peter Niebert, editors, *FORMATS*, volume 2791 of *Lecture Notes in Computer Science*, pages 34–45. Springer, 2003.

[ZST12] Aditya Zutshi, Sriram Sankaranarayanan, and Ashish Tiwari. Timed Relational Abstractions for Sampled Data Control Systems. In Madhusudan and Seshia [MS12], pages 343–361.

Index

Appendix A

Appendix

A.1 Additional Proofs

Proof. Lemma 2 If $enc(\langle m, \phi \rangle, \overline{k})$ is satisfiable then there exists a path π consistent with $\langle m, \phi \rangle$ such that for all $1 \leq i \leq n$, for all $1 \leq j \leq |\sigma_i|$, $|loc_j(prj(i, \pi))|$ is $k_i[j]$.

π can be extended to a path π' as follows. For all $1 \leq i \leq n$, $0 \leq j \leq |\sigma_i|$:

- $pre_j(prj(\pi, i)) = pre_j(prj(\pi', i))$,
- $post_j(prj(\pi, i)) = post_j(prj(\pi', i))$,
- $loc_j(prj(\pi', i)) := loc_j(prj(\pi, i)); \varepsilon = \text{S}; s_1; \ldots; \varepsilon = \text{S}; s_{k_i'[j] - k_i[j]}$, where for $1 \leq z \leq k_i'[j] - k_i[j]$, s_z is equal to the last state of $loc_j(prj(\pi, i))$. $loc_j(prj(\pi', i))$ is the concatenation of the j-th local sequence of π ($loc_j(prj(\pi, i))$) with a sequence of stutter actions (i.e. $\varepsilon = \text{S}$).

Thus, π' extends all the local sequences of π by stuttering actions S.

Since stutter does not change the state reached by the system, $\pi' \models enc(\langle m, \phi \rangle, \overline{k}')$. $\diamond$

Proof. Lemma 3 For all $1 \leq i \leq n$ and $j = 0$ if $kind_i[j]$ is unsatisfiable, $\pi \models enc(\langle \overline{m}_i[0], \overline{\phi}_i[0] \rangle, \overline{k})$ and $|loc_0(prj(i, \pi))| > k_i[j]$. In this case

$enc(\langle\overline{m}_i[0], \overline{\phi}_i[0]\rangle, \overline{k})$ encodes the local segment $lsg(\sigma_i[0])$ and $kind_i[0]$ is unsatisfiable. Thus, it does not exist a simple path $loc_0(prj(i, \pi))$ longer than $lsg(\sigma_i[0])$.

Consider the local segment $lsg(\sigma_i[j])$ and suppose that the lemma holds for all l, h such that $lsg(\sigma_l[h]) <_m lsg(\sigma_i[j])$ (i.e. the theorem holds for all the local segment which are found "before" in the partial order defined by $<_m$). Suppose that the bounds $\overline{k}'$ are the same as $\overline{k}'$, except for $k_i'[j]$ which is $k_i[j] + 1$. For every path π such that $\pi \models enc(\langle\overline{m}_i[j], \overline{\phi}_i[j]\rangle, \overline{k}')$, we have that $loc_j(prj(i, \pi))$ is not a simple path, since $kind_i[j]$ is unsatisfiable.

By induction, the lemma holds for all $loc_j(prj(i, \pi))$. $\diamond$

A.2 Relational abstraction

Consider the linear system $\dot{\vec{x}} = A\vec{x} + b$.

The system of differential equations is modified partitioning e the variables $\vec{x}$ into $\vec{y}$ and $\vec{z}$ such that:[1]

$$\begin{bmatrix} \dot{\vec{y}} \\ \dot{\vec{z}} \end{bmatrix} = \begin{bmatrix} A_1 & A_2 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \vec{y} \\ \vec{z} \end{bmatrix} + \begin{bmatrix} \vec{b_1} \\ \vec{b_2} \end{bmatrix}$$

Where $A_1 \in \mathbb{R}^{n_1 \times n_1}, A_2 \in \mathbb{R}^{n_2 \times n_2}$, $\vec{b_1} \in \mathbb{R}^{n_1}, \vec{b_2} \in \mathbb{R}^{n_2}$ and $n = n_1 + n_2$.

A.2.1 Real eigenvalues

Suppose $\vec{c}$ is a left eigenvector of A_1 corresponding to some real eigenvalue λ (i.e. $\vec{c}^T A = \lambda \vec{c}^T$). Then we define the linear expression $p(x)$ as follows:

$$\begin{aligned} p(\vec{x}) &:= \vec{c}^T \vec{y} + \vec{d}^T \vec{z} + e \\ \vec{d}^T &:= \tfrac{\vec{c}^T A_2}{\lambda} \\ e &:= \tfrac{\vec{c}^T \vec{b_1} + \vec{d}^T \vec{b_2}}{\lambda} \end{aligned}$$

[1]Note that if there are no variables with constant derivative the dimension of $\vec{z}$ is 0, which is a simpler special case.

Thus, $p(\vec{x}) = \vec{c}^T\vec{y} + \vec{d}^T + \vec{z} + e$ and it is such that:

$$\begin{aligned}
\tfrac{dp(\vec{x})}{dt} &= \tfrac{d(\vec{c}^T\vec{y}+\vec{d}^T\vec{z}+e)}{dt} \\
&= \vec{c}^T\tfrac{d\vec{y}}{dt} + \vec{d}^T\tfrac{d\vec{z}}{dt} \\
&= \vec{c}^T(A_1\vec{y} + A_2\vec{z} + \vec{b_1}) + \vec{d}^T b2 \\
&= \lambda\vec{c}^T\vec{y} + \vec{c}^T A_2\vec{z} + \vec{c}^T\vec{b_1} + \vec{d}^T b2 \\
&= \lambda(\vec{c}^T\vec{y} + \tfrac{\vec{c}^T A_2\vec{z}}{\lambda} + \tfrac{\vec{c}^T\vec{b_1}+\vec{d}^T b2}{\lambda}) \\
&= \lambda(\vec{c}^T\vec{y} + \vec{d}^T\vec{z} + e) \\
&= \lambda p(\vec{x})
\end{aligned}$$

Thus, also for the general case when $n_2 > 0$ it holds that:

$$p(\vec{x}(t)) = p(\vec{x}(0))e^{\lambda t} \tag{A.1}$$

A.2.2 Complex eigenvalues

Let $\vec{v} := \vec{c} + \imath\vec{d}$ be a left eigenvector of A corresponding to the complex eigenvalue $\lambda := a + \imath b$:

$$(\vec{d}^T + \imath\vec{e}^T)A = (a + \imath b)(\vec{d}^T + \imath\vec{e}^T)$$

Equating the real and imaginary parts in the above equation, we get:

$$\vec{c}^T A_1 = a\vec{c}^T - b\vec{d}^T \qquad \vec{d}^T A_1 = b\vec{c}^T + a\vec{d}^T \tag{A.2}$$

Consider the two expressions:

$$p(\vec{x}) = \vec{c}^T\vec{y} + \vec{c_1}^T\vec{z} + e_1 \qquad q(\vec{x}) = \vec{d}^T\vec{y} + \vec{c_2}^T\vec{z} + e_2 \tag{A.3}$$

where $\vec{c_1} \in \mathbb{R}^n, \vec{c_2} \in \mathbb{R}^n, e_1 \in \mathbb{R}$ and $e_2 \in \mathbb{R}$.

We compute $\vec{c_1}, \vec{c_2}, e_1$ and e_2 to satisfy:

$$\frac{d}{dt}\vec{p} = ap - bq \qquad \frac{d}{dt}\vec{q} = bp + aq \tag{A.4}$$

First, let's compute the time derivative of p and q:

$$\frac{d}{dt}\vec{p} = \vec{c}^T A_1 \vec{y} + \vec{c}^T A_2 \vec{z} + \vec{c}^T \vec{b_1} + \vec{c_1}^T b2 \tag{A.5}$$

$$\frac{d}{dt}\vec{q} = \vec{d}^T A_1 \vec{y} + \vec{d}^T A_2 \vec{z} + \vec{d}^T \vec{b_2} + \vec{c_2}^T b2 \tag{A.6}$$

Equating $\vec{y}$, $\vec{z}$ and the constant terms of Equations A.4 and Equations A.5,A.6 we get the following equations among the unknown parameters $\vec{c_1}, \vec{c_2}, e_1$ and e_2:

$$\vec{c}^T A_2 = a\vec{c_1}^T - b\vec{c_2}^T \qquad \vec{d}^T A_2 = a\vec{c_2}^T + b\vec{c_2}^T \tag{A.7}$$

$$\vec{c}^T \vec{b_1} + \vec{c_1}^T b_2 = ae_1 - be_2 \qquad \vec{d}^T \vec{b_1} + \vec{c_2}^T b_2 = ae_2 + be_1 \tag{A.8}$$

From Equations A.7 we compute the values of $\vec{c_1}^T$ and $\vec{c_2}^T$:

$$\vec{c_1}^T = \frac{a\vec{d}^T - b\vec{c}^T}{a^2+b^2} A_2 \tag{A.9}$$

$$\vec{c_2}^T = \frac{a\vec{c}^T + b\vec{d}^T}{a^2+b^2} A_2 \tag{A.10}$$

Now, define $k_1 = \vec{c}^T \vec{b_1} + \vec{c_1}^T b_2$ and $k_2 = \vec{d}^T \vec{b_1} + \vec{c_2}^T b_2$. From Equations A.8, we compute the values of e_1 and e_2:

$$e_1 = \frac{ak_1 + bk_2}{a^2+b^2} \tag{A.11}$$

$$e_2 = \frac{ak_2 - bk_1}{a^2+b^2} \tag{A.12}$$

Now, note that the relationship between p and q is the same as the one presented in Section 5.3.4. Thus, all the results obtained in the simpler case $Flow(q) = A\vec{x}$ hold also for the case $Flow(q) = A\vec{x} + \vec{b}$.