INSTITUT POLYTECHNIQUE DE PARIS

ÉCOLE POLYTECHNIQUE

IP PARIS

Thèse de Habilitation à Diriger des Recherches

Abstraction Techniques for Symbolic Model Checking of Infinite-state Discrete and Continuous Systems

Habilitation à Diriger des Recherches de l'Institut Polytechnique de Paris
préparée à École Polytechnique

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat: Informatique, données, intelligence artificielle

Thèse présentée et soutenue à Palaiseau, le 13/05/2024, par

Sergio Mover

Composition du Jury :

Benedikt Bollig Directeur de Recherche, ENS Paris-Saclay, Université Paris-Saclay, CNRS	Rapporteur
Sylvain Conchon Professeur, Université Paris-Saclay	Examinateur
Martin Fränzle Professor, Carl von Ossietzky Universität Oldenburg	Rapporteur
Goran Frehse Professeur, ENSTA Paris	Rapporteur
Antoine Miné Professeur, LIP6, Sorbonne Université & CNRS	Examinateur
Marc Pouzet Professeur, École Normale Supérieure	Examinateur
Fabio Somenzi Professor, University Of Colorado Boulder	Examinateur

Abstract

The design of safety- and mission-critical software systems (used in, e.g., avionics, automotive, medical devices, …) requires the development of automated tools, such as Model Checking, for analyzing all the possible executions and for determining if the system is correct. An existing problem is to extend model checking algorithms to analyze efficiently "expressive" systems that, for example, can have an infinite state space (e.g., software working on real or integer numbers) or mix discrete-time and continuous-time (e.g., control software interacting with the physical environment). In this habilitation thesis, we summarize some of the approaches the author contributed to solving the verification problem for different families of infinite-state systems. First, we focus on the Verification Modulo Theory (VMT) problem, that is model checking infinite-state transition systems expressed with first-order theories. Then, we focus on the problem of verifying hybrid systems, where a discrete-time system interacts with a continuous-time system expressed with Ordinary Differential Equations (ODEs). In both cases, we consider the invariant and the liveness model checking problems.

We first describe the symbolic algorithm IC3-IA that solves the VMT problem using abstraction to cope with the infinite-state space challenge. The algorithm is general enough to work for a wide set of first-order theories and addresses the challenge of exploring efficiently a space of exponentially large abstract transition systems (i.e., a Counter-Example Guided Abstraction Refinement loop). We also extend such invariant verification algorithm to prove liveness properties via a liveness-to-safety reduction.

Then, we investigate the use of abstraction as a tool to obtain a purely discrete transition system to over-approximate a hybrid system expressed with non-linear ODEs (i.e., a reduction to a VMT problem). We first show how to efficiently model check qualitative abstractions (i.e., abstractions defined by the signs of a set of polynomials), tackling the exponential state-space explosion with similar ideas to IC3-IA, and then we show how to scale the computation of relational abstractions (i.e., an approximation of the continuous system's trajectories) for non-linear dynamical systems exploiting the variables dependencies in the ODEs.

Keywords
Model Checking, Satisfiability Modulo Theories, Verification Modulo Theory, Infinite-State Systems, Hybrid Systems, Abstraction, CEGAR

To Heather and Claire,

Acknowledgments

I would first like to thank Benedikt Bollig, Sylvain Conchon, Martin Fränzle, Goran Frehse, Antoine Miné, Marc Pouzet, and Fabio Somenzi for kindly accepting to evaluate this work and serving in the habilitation jury. In particular, I'd like to thank Benedikt Bollig, Martin Fränzle, and Goran Frehse for serving as reviewers and providing a thorough evaluation and feedback on this manuscript.

I would like to thank Alessandro Cimatti, Stefano Tonetta, Bor-Yuh Evan Chang, and Sriram Sankaranarayanan that, in different capacities, kindly provided me with guidance and advice in the past years. I would like to thank all the other co-authors of the works presented in this manuscript (listed in alphabetic order): Xin Chen, Jackub Daniel, Alberto Griggio, Ahmed Irfan, Marco Roveri, and Sriram Sankaranarayanan. I'd also thank the other collaborators I had the pleasure to work with over the years and, in particular, Shawn Meier, Sao Mai Nguyen, Mirko Sessa, and Mehdi Zadem.

A major thank goes to my colleagues from the Cosynus team at LIX (Laboratoire d'informatique de l'École Polytechnique): Constantin Enea, Uli Farhenberg (now at EPITA), Eric Goubault, Samuel Mimram, Emmanuel Haucourt, and Sylvie Putot. In particular, Eric and Sylvie for helping me several times during the years, and Samuel for pushing me to complete this manuscript. Finally, I'd like to thank Benjamin Doerr for all the guidance in figuring out and completing the habilitation process at IP Paris.

Contents

Chapter 1

Introduction

1.1 Context and Motivations

Modern software systems are at the core of safety- and mission-critical applications in different domains (e.g., avionics, automotive, medical devices, ...). Designing such systems requires automated tools for checking that they are correct and secure. *Model Checking* [112] algorithms automatically analyze all the possible executions of a system to determine if it satisfies a specification expressing the intended system's behavior.

One of the main challenges in model checking, and formal methods in general, is to analyze *"complex"* systems. In general, the model checking problem becomes more complex and even undecidable when dealing with *more expressive* systems. For example, embedded software requires to efficiently reason on large bit vectors and floating-point numbers, and verifying high-level models (e.g., from model-based design [121, 54]) often requires to reason on Integer and Real numbers. To make things worse, when the software interacts with the physical environment, the underlying system becomes *hybrid* and the *discrete time* software interacts with processes evolving in *continuous time* (e.g., timed automata [4], hybrid automata [8]). This last scenario is typical for control systems where the physical environment is modeled with differential equations. In this thesis, we focus on model checking invariant and liveness properties (e.g., obtained from *Linear Temporal Logic* (LTL) [6] specifications) for two expressive classes of infinite state systems: (i) transition systems represented with first-order theories, and (ii) hybrid systems mixing discrete and continuous dynamics.

Verification Modulo Theories. We will first focus on the verification problems for transition systems expressed with first-order theories, called *Verification Modulo Theories* (VMT) [**J7**] by analogy with the Satisfiability Modulo Theory (SMT($\mathcal{T}$)) problem. First-order theories provide a generic framework to express different systems. For example, the theory of Bit-Vectors (BV) can express embedded C programs (i.e., programs with bit-vector operations but without dynamic memory and recursion), the theory of Difference Logic (DL) can express timed automata, and the theory of Linear Real Arithmetic (LRA) can express Linear Hybrid Automata (i.e., automata with piece-wise constant dynamics). In practice, the use of theories allows us to express, symbolically, a transition system capturing the semantic of different classes of systems, including programs and other systems' models (e.g., timed automata). Developing efficient verification algorithms for such symbolic transition systems allows us to model check, within a single framework, a wide family of target systems. What makes the development of such verification algorithms possible is the continuous improvement in efficient *Satisfiability Modulo Theory* (SMT) [136] solvers, which decide the satisfiability problem of formulas expressed using first-order theories, and further solve other automated reasoning problems (e.g., computing interpolants).

Verification of Non-Linear Hybrid Systems. We will then focus on the model checking problem for *hybrid systems*, where a *discrete* software monitors and controls the physical environment that evolves *continuously* in time. Hybrid systems often express the continuous dynamic of the systems with a system of *Ordinary Differential Equations* (ODEs), which defines how the derivative of each state variable changes over time in function of the other quantities (i.e., state variables, input variables, time elapsed, ...). While the verification problems for hybrid systems is undecidable even for relatively simple dynamics (e.g., see [16, 15]), what is difficult even when designing incomplete verification algorithms is to cope with *non-linear continuous dynamics* where the equations in the system of ODEs are polynomials or transcendental functions. In this thesis, we mainly consider the problem of *proving* properties for hybrid systems, also in the presence of non-linear dynamics.

1.2 Challenges and Contributions

In this section we present the challenges in solving the VMT and the hybrid systems verification problems and the main contributions of this thesis.

1.2.1 Verification Modulo Theories (VMT)

VMT can be seen as the extension of the verification problem for symbolic *finite state* transition systems expressed with propositional logic to *infinite state* transition systems (i.e., extend the model checking algorithms from transition system expressed with propositional logic to transition system expressed with first-order theories). The state of the art in model checking symbolic finite state transition systems, mostly explored in the hardware verification domain, are *SAT-based* algorithms (e.g., BMC [12], IC3 [55], interpolation-based model checking [25], k-induction [17], ...) that repeatedly query a SAT solver. One of the hopes to tackle the VMT problem was to modify the existing, efficient, SAT-based model checking algorithms using, instead, an SMT solver. The idea is appealing since, after all, SMT solvers already extends SAT solvers to cope with first-order theories.

However, extending the algorithms from the propositional to the first-order logic settings is challenging, mainly because the systems have an infinite number of states. In fact, to be effective, several verification algorithms require the assumption that the system is finite state and, for this reason, a naïve instantiation of the algorithms swapping a SAT for an SMT solver would not be effective. For example, to prove an invariant property k-induction [17] checks the non-existence of "simple paths" longer than a positive constant k (i.e., paths that do not contain loops); such check may not succeed when the states are infinite since the number of simple-paths in the system can also be infinite. IC3 also has similar assumptions and would be ineffective in finding an inductive invariant (e.g., see [64]). Similarly, SAT-based algorithms for proving liveness properties (e.g., L2S [20], k-liveness [65]) assume omega paths (i.e., infinite paths) to be represented as lasso-shaped paths (i.e., paths composed of a finite prefix and a loop) and such assumption does not hold anymore when the system has an infinite number of states [30].

An obvious and common approach to deal with an infinite number of states is abstraction [3, 5], where an infinite set of states is abstracted with a finite number of states. Key to model checking using abstractions is the *Counter-Example Guided Abstraction Refinement* (CEGAR) loop [24] that allows a verification algorithm to automatically and gradually find a precise enough abstraction to prove a property, using spurious counterexamples at every iteration to guide the abstraction refinement. While such framework has been successfully applied to different systems, in particular software (e.g., see [18, 21]), the CEGAR loop is often computationally prohibitive, in particular when using Predicate Abstraction [9] and when

the system under analysis does not have a control flow structure, as in the case of symbolic transition systems. The first research question we answer in this thesis is:

How can we *effectively* use abstractions to model check invariant and liveness properties of infinte-state symbolic transition systems?

Contributions [C12, J5, C17, J7]. The main contribution to tackle the VMT problem for invariant properties is IC3IA [C12, J5]. IC3IA adapts the IC3 [55] model checking algorithm to work on transition systems expressed with first-order theories using abstraction. From a high-level point of view, IC3IA performs a standard CEGAR loop verifying different predicate abstractions of the system. However, IC3IA does not explicitly compute the abstract, finite state transition system at each CEGAR iteration, avoiding the main bottleneck of predicate abstraction computation. Instead, IC3IA checks the relative induction among set of *abstract states*, the main operation of IC3, with a single satisfiability check (using *Implicit Abstraction* [44]). While such approach is conceptually "simple", it provides the following advantages.

1. It is *theory agnostic*: IC3IA does not require a novel algorithm for each new theory (differently from [67]). Instead, the only requirements are the decidability of the satisfiability problem for a theory and the existence of an interpolation procedure for the abstraction refinement.
2. It is *incremental*: the algorithm does not restart from scratch after every refinement, but instead retains the reachable state's approximations computed so far (i.e., the IC3 frames).
3. It is *efficient*: the algorithm scales in verifying abstraction with a high number of predicates (e.g., in the order of hundreds).

The second contribution is a verification algorithm for liveness properties [C17]. The algorithm implements a *liveness to safety* reduction [20] for infinite state systems. The idea of the liveness to safety reduction is to cast a liveness verification problem as an invariant verification problem adding additional states to the transition system to recognize loops. However, such reduction ignores non lasso-shaped paths, so it cannot prove that a liveness property holds for an infinite-state system. We solve such problem applying the liveness to safety reduction to a finite state abstraction of the system, similarly in spirit to [30]. Clearly, finding an abstract lasso shaped path does not mean the system does not satisfy a liveness property. In practice, when the algorithm finds an abstract lasso shaped counterexample, it has to prove that all its *concretizations* are finite (i.e., they terminate [76]). The termination proof provides a well-founded relation that "rules out" such concrete paths. The algorithm works in a CEGAR fashion: it refines both the abstraction predicates and a well-founded relation. Such algorithm is built on top of IC3IA and share with it several advantages (e.g., incrementality). Both the algorithms have been implemented in the *nuXmv* [C11] model checker.

The bottom half of Figure 1.1 shows the verification flow for the VMT problem: (i) the IC3IA algorithm takes as input the infinite state transition system expressed with theories S and an invariant property P_S and decide if $S \models P_S$ (the algorithm may also not terminate as the problem is undecidable). (ii) the L2S-IA algorithm takes as input a transition system S and a liveness property, here expressed as $\mathbf{FG}\ \neg f$, and decide if $S \models \mathbf{FG}\ \neg f$.

1.2.2 Verification of Hybrid Systems with Discrete Abstractions

We focus on the problem of *proving* if a property, in particular an invariant or a LTL specification, holds for a hybrid system. What is challenging about analyzing hybrid systems,

in particular for non-linear dynamics, is that a system of differential equations usually does not have an explicit solution (i.e., a function that computes the state the system reaches after some amount of time elapses). An existing approach to tackle such problem consists of reducing the hybrid system model checking problem to the model checking problem of a discrete system (e.g., [10, 15, 62, 43]). The core of such reduction is the computation of a *discrete abstraction* of a hybrid system that is then amenable to model checking. For some dynamics (e.g., timed automata, piece-wise constant hybrid automata, o-minimal hybrid automata) one can compute abstractions (e.g.,see [15]) that preserve both invariant and LTL properties. Instead, for more "complex" dynamics such discretizations usually produce an abstraction that over-approximates the system behavior (e.g., [62, 102, **C10**]). What is interesting is that the discrete abstract system can be analyzed using efficient VMT algorithms.

In this thesis, we tackle the following problems related to the computation and verification of discrete abstractions of a hybrid system: (A) efficient verification of semi-algebraic abstractions [43, 102] for non-linear systems; (B) computation of relational abstractions [62] for non-linear systems; and (C) verifying LTL properties using a discretizations of a hybrid systems. In details:

(A) A semi-algebraic abstraction [102] is a qualitative abstraction [43] that partitions the state space according to the sign of a list of polynomials, similarly to a predicate abstraction. The abstraction has a finite number of states that can be computed for a system of differential equations defined with polynomials (i.e., the right-hand side of each equation is a polynomial) and can then be easily model checked (e.g., via an explicit-state reachability analysis). However, computing the abstraction is challenging since the number of abstract states is, in the worst case, exponential in the number of polynomials.

(B) A relational abstraction [62] represents all the possible trajectories of a dynamical system with a relation, which sometimes is expressed using first-order theories. If a state s reaches a state s' in the dynamical system then the pair (s, s') is included in the relation. We can use such relation in a discrete transition system to over-approximate the continuous dynamic. Computing a useful relational abstraction is hard, in particular for a non-linear system. First, we cannot use the properties of linear systems, like the existence of a closed form solution of the ODEs (e.g., see [71]). Instead, Taylor model based flow-pipe construction [89] can compute a relational abstraction for a non-linear system on a bounded domain. However, there is a tradeoff between the precision of the abstraction and the scalability of such computation.

(C) Reducing a liveness model checking problem to an invariant model checking problem is convenient, since it allows us to reuse the existing and efficient invariant model checking algorithms. However, we need additional care when applying such paradigm to the verification of continuous time and hybrid systems. We focus on the `k-liveness` verification algorithm [65]. In a nutshell, `k-liveness` tries to prove that $\mathbf{FG}\ \neg f$, for a set of states f, by proving that the number of times the system visits a state in $\neg f$ is bounded. If that's the case, then the system visits f infinitely often. We can use `k-liveness` to prove a liveness property for hybrid systems: we first obtain a discrete transition systems S abstracting H, and then we use `k-liveness` to prove that $S \models \mathbf{FG}\ \neg f$. However, `k-liveness` fails as soon as H contains *Zeno paths*, infinite paths such that the total time elapsed in the path does not diverge. A Zeno path represents an unrealistic path where the system executes an infinite number of discrete transitions in a finite amount of time and that should not be considered in the system's semantic. A naïve application of `k-liveness` would not be able to prove $S \models \mathbf{FG}\ \neg f$, always finding a Zeno path satisfying $\neg f$. We tackle the problem of avoiding such paths when proving liveness properties.

To summarize, the research questions we explore in this thesis are:

How can we discretize non-linear dynamics to reduce the verification problem for a hybrid system to the verification problem of a discrete one?

How can we use a discretization to verify liveness properties?

Contributions [C13, J6, C25]. In [C25], we tackle the problem of verifying a semi-algebraic abstractions for a non-linear dynamical system avoiding the explicit enumeration of the abstract states (challenge **(A)**). We apply the same idea of implicit abstraction [44] we also used in the IC3IA algorithm, following the intuition that a semi-algebraic abstraction is very similar to a predicate abstraction. What differs in the computation of the semi-algebraic abstraction is that it uses a decision procedure for checking differential invariants [60], instead of the system's continuous non-linear dynamics (which we cannot compute easily for an unbounded time horizon). Our solution encodes symbolically the abstract transition relation using a formula that has *linear*, instead of exponential, size in the number of abstraction polynomials. The approach carries the same advantages of implicit abstraction since it avoids the up-front exponential blowup of the abstraction computation. We implemented the symbolic encoding in the *Sabbath* tool, using the *nuXmv* model checker as verification backend (in practice, we use the model checking algorithm for transition systems expressed with Non-linear Real Arithmetic formulas [105] that is based on the IC3IA algorithm). [1]

In [J6], we tackle the relational abstraction computation problem for non-linear systems via Taylor model based flow-pipe construction [89]. We tackle the challenge of computing a precise abstraction while limiting the number of subdivisions of the state space (challenge **(B)**). In our solution, we compute the relational abstraction compositionally, partitioning the non-linear system of ODEs according to the dependencies of the variables in the system. This algorithm has been implemented using *FLOW** [74] for the Taylor model computation and *nuXmv* as verification backend.

Finally, in [C13] we provide the K-Zeno algorithm for proving LTL properties for hybrid systems using the `k-liveness` algorithm. In [C13], we remove the Zeno paths applying the `k-liveness` algorithm (challenge **(C)**) to the system composed with a monitor automaton that forces the repeated occurrences of the condition f to be separated by a "sufficient" amount of time. While such amount is a positive constant for timed automata, in general it is a parameter that depends on the automaton's continuous dynamics, guards, and invariants. We provide an automaton construction that determines such bound and that is complete (i.e., if the liveness property holds and the backend invariant verification algorithm always terminates) for a subclass of hybrid automata . We implemented K-Zeno, together with other discretization algorithms for piece-wise constant and linear systems, in the *HyCOMP* [C15] model checker.

The top half of Figure 1.1 shows the verification flow using discretization for hybrid automata:

- The discretization blocks (enclosed in the purple dashed block shown at the top-left of the figure) applies a different discretization to the hybrid automaton H depending on H's dynamics. The result of the discretization is an infinite state transition system S.
- To prove the hybrid automaton H satisfies the invariant property P_H we prove that $S \models P_S$ for an invariant property P_S (e.g., with IC3IA, shown in the bottom-left part of Figure 1.1). While we obtain the property P_S from P_H depending on the specific abstraction, we don't show this graphically in Figure 1.1 to reduce the clutter.

[1] *Sabbath* is available online `https://github.com/cosynus-lix/sabbath`.

- To prove that $H \models \mathbf{FG}\ \neg f$ the K-Zeno algorithm computes an additional monitor automaton Z_β using the hybrid systems H (top-right part of Figure 1.1). Z_β is also a discrete symbolic transition system. Then, we use the `k-liveness` algorithm (bottom-right part of Figure 1.1) to prove that $S_H \times Z_\beta \models \mathbf{FG}\ \neg f_\beta$, where f_β is the acceptance condition of the automaton Z_β that is true if f holds after "enough time" elapsed from the previous occurrence of f. If such check succeed, we conclude that $H \models \mathbf{FG}\ \neg f$.

1.3 Scope of the Thesis and Thesis Structure

This thesis presents coherently contributions to the model checking problem of infinite-state transition systems [C12, J5, C17, J7] and hybrid systems [C13, J6, C25], and does not provide novel contributions. Also, the thesis presents in depth a subset of the works performed after the Ph.D. degree that can be presented uniformly. [2] The thesis provides a high-level overview (in Chapter 5) of the other major contributions to the following research topics:

1. Analysis of Switched Kirchhoff Networks [C16, C18, C19];
2. Verification [C23, J8] and specification synthesis [C20, C21] of event-driven programs; and
3. Learning abstractions in goal-conditioned Hierarchical Reinforcement Learning [C27, C28].

Also, the thesis does not present other publications [C3, C1, J1, J4, C14, C24, C26]. In the interest of brevity, we do not include theorems (which usually show the soundness of the algorithms), their proofs, experimental evaluations, and the details of the tools' implementation (e.g., *nuXmv* [C11] and *HyCOMP* [C15]). We refer to the corresponding publications for such information.

Experience in students' advising. Several of my research contributions have been obtained advising, in different capacities, Ph.D. and undergraduate students. In chronological order, I advised the following students:

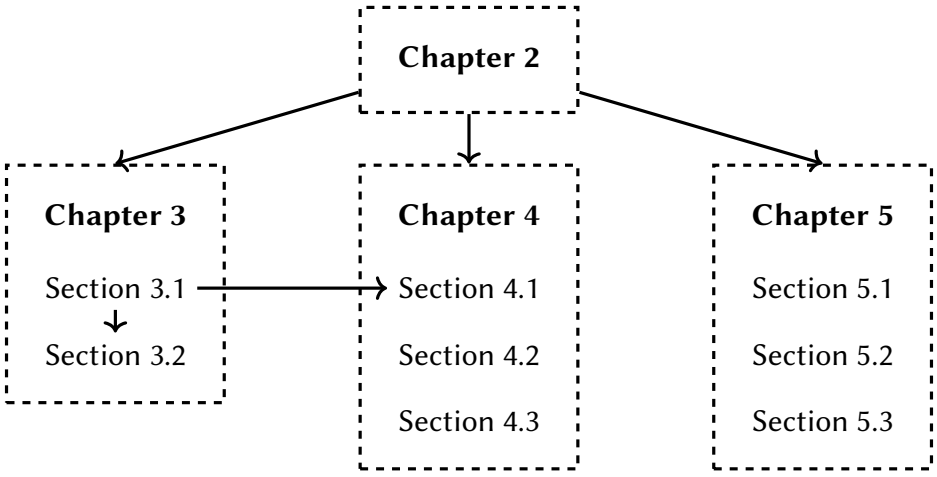
- Qiang Wang: I co-advised Qiang (Ph.D. candidate from EPFL, graduated in 2017) with Alessandro Cimatti (Fondazione Bruno Kessler) during a summer internship in 2014. We published a paper on model checking BIP models [C14].
- Rhys Braginton Pettee Olsen: I co-advised Rhys (Undergraduate student at the University of Colorado Boulder, graduated in 2018) with Sriram Sankaranarayanan (University of Colorado Boulder) while Rhys worked as research assistant in 2016 and 2017. We published a paper on mining API specifications from a large corpus of programs [C20].
- Mirko Sessa: I co-advised Mirko (Ph.D. candidate at the University of Trento, graduated in 2019) with Alessandro Cimatti (Fondazione Bruno Kessler) from 2015 to 2019. With Mirko, we worked on the problem Formal Analysis of Switched Kirchhoff Networks [C16, C18, C19].
- Shawn Meier: I am co-advising Shawn (Ph.D. candidate at the University of Colorado Boulder, plan to graduate in 2024) with Bor-Yuh Evan Chang (University of Colorado Boulder). Shawn is working on the formalization, verification, and synthesis for event-driven programs [C21, C23, J8].

[2] The author's Ph.D. thesis [T1] included the following publications: verification of distributed hybrid systems [C2, C5, C4, C6, J2], discretization of hybrid systems [C8, C7, J3, C10], and parameter synthesis for infinite-state systems [C9].

- Mehdi Zadem: I am co-advising Mehdi (Ph.D. candidate at the Institut Polytechnique de Paris, plan to graduate in 2024) with Sao Mai Nguyen (ENSTA Paris). Mehdi is working on abstractions in Hierarchical Reinforcement Learning [**C27**, **C28**].

Structure of the Thesis. The thesis has the following structure and can be read following the graph in the figure below:

- Chapter 2 introduces the common notation used in the thesis, defines the VMT problem and the Hybrid Automata verification problem.
- Chapter 3 first presents IC3ia (Section 3.1) and then the L2S-IA (Section 3.2) algorithm for proving liveness properties for infinite-state transition systems.
- Chapter 4 presents Implicit Semi-Algebraic Abstraction (Section 4.1), Compositional Relational (Section 4.2), and finally the K-Zeno algorithm (Section 4.3).
- Chapter 5 summarizes the other main research areas I contributed to: Formal Analysis of Switched Kirchhoff Networks (Section 5.1), Event-Driven Program Verification and Synthesis (Section 5.2), and Abstractions in Hierarchical Reinforcement Learning (Section 5.3).



All the chapters depend (solid arrow) on the preliminaries (Chapter 2). Inside each chapter, Section 3.2 (Liveness-to-Safety reduction via Implicit Abstraction) depends on Section 3.1 (IC3ia). Across chapters, Section 4.1 (Implicit Semi-Algebraic Abstraction) depends on Section 3.1 that defines implicit abstraction.

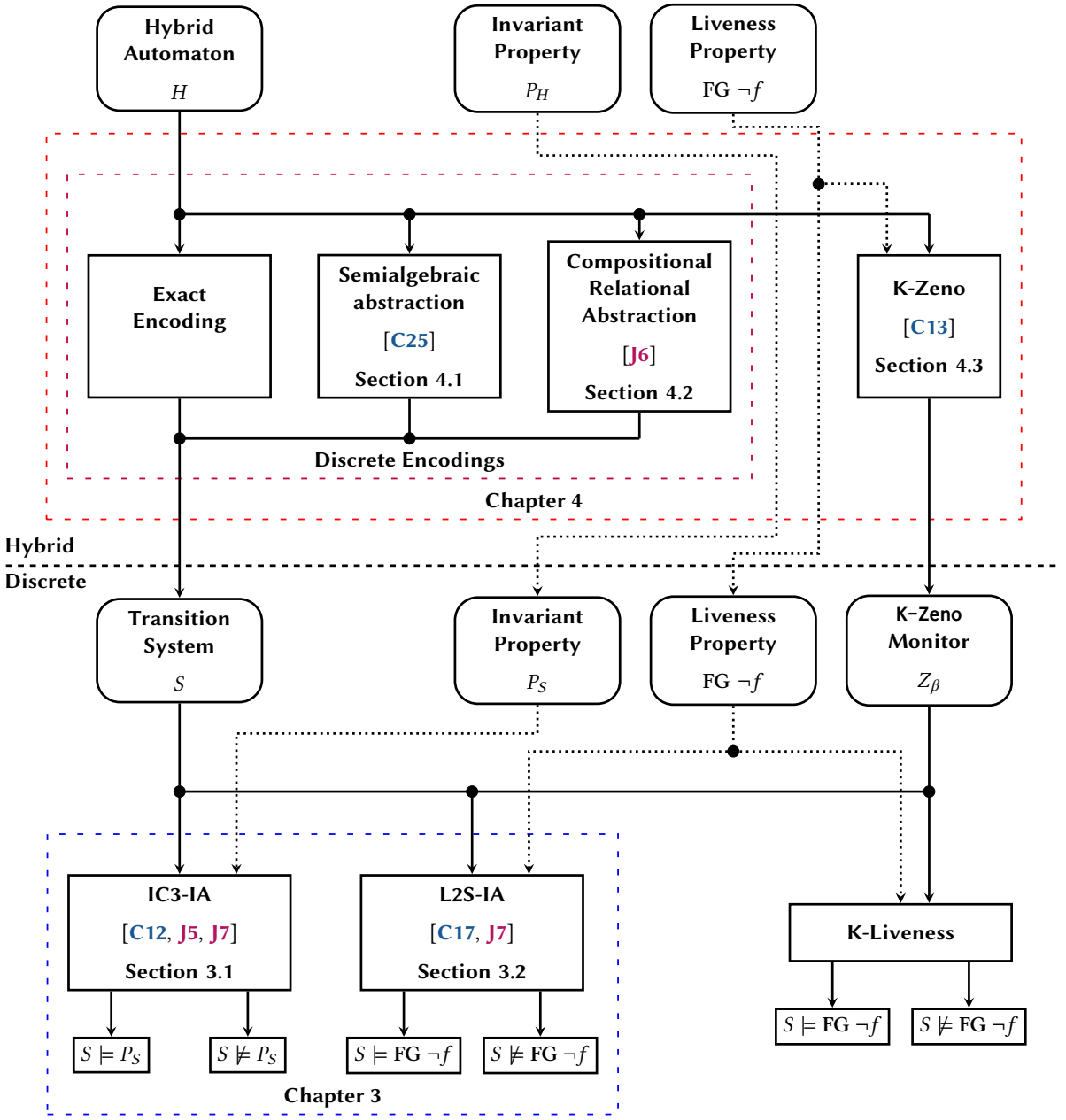


Figure 1.1: **Thesis contributions and organization.**The *lower half* of the image shows the *Verification Modulo Theories* (VMT) algorithms. The blue, dashed box encloses the contributions of this thesis for VMT: given a transition system S (represented with first order theories), the **IC3-IA** solves the invariant verification problem (i.e., $S \models P_S$), and the **L2S-IA** algorithm solves the liveness verification problem (i.e., $S \models$ **FG** $\neg f$). The *upper half* of the image shows the verification algorithms for hybrid systems and the contributions of this thesis in the red dashed box. The smaller purple dashed box contains the discretization techniques encoding a hybrid automata as a discrete system (i.e., a transition system S). The thesis describes two abstraction techniques for non-linear hybrid systems. While the discrete abstractions can be used to prove invariant and liveness properties, the K-Zeno algorithm takes care of ignoring Zeno paths that prevent the algorithm's termination. Boxes without a reference (e.g., the `k-liveness` algorithm) are not a contribution of this thesis.

Chapter 2

Background on Safety and LTL Verification

We briefly introduce the invariant and liveness verification problems for discrete symbolic transition systems expressed with theories and for both dynamical and hybrid systems.

2.1 Satisfiability Modulo Theories

In the following, we use standard notation from first-order logic (e.g., see [36]). We assume to be given a signature Σ of function and predicate symbols. A 0-ary function symbol is called a *constant*. A Σ-*term* is a first-order term built out of function symbols and variables. If $t_1, \ldots, t_n$ are Σ-terms and p is a predicate symbol, then $p(t_1, \ldots, t_n)$ is a Σ-*atom*. A Σ-*formula* ϕ is built in the usual way out of the universal and existential quantifiers, Boolean connectives, and Σ-atoms. When Σ is implicit, we omit it and just talk about terms, atoms, and formulas. A *literal* is either an atom or its negation. We call a formula *quantifier-free* if it does not contain quantifiers, and *ground* if it does not contain free variables. A *clause* is a disjunction of literals. A formula is said to be in *conjunctive normal form* (CNF) if it is a conjunction of clauses. For every non-CNF formula φ, an equisatisfiable CNF formula ψ can be generated in polynomial time [2]. We assume the usual first-order notions of interpretation, satisfiability, validity, logical consequence, and theory. We write $\Gamma \models \phi$ to denote that the formula ϕ is a logical consequence of the (possibly infinite) set of formulas Γ. A *first-order theory*, $\mathcal{T}$, is a set of first-order sentences. A structure $\mathcal{A}$ is a model of a theory $\mathcal{T}$ if $\mathcal{A}$ satisfies every sentence in $\mathcal{T}$. A formula is *satisfiable in* $\mathcal{T}$ if it is satisfiable in a model of $\mathcal{T}$.

Moreover, following the terminology of the SAT and SMT communities, we refer to predicates of arity zero as *propositional variables*, and to uninterpreted constants as *theory variables*. Finally, if a formula φ is satisfiable, we call a *model* of φ any assignment μ to (possibly a subset of) the variables of φ and interpretation of symbols $\langle \mathcal{M}, \mu \rangle$ which make the formula true, and we denote this with $\mu \models \varphi$. If μ is a model and x is a variable, we write $\mu[x]$ for the value of x in μ.

Given a first-order theory $\mathcal{T}$, an *SMT solver* for $\mathcal{T}$, $\mathsf{SMT}(\mathcal{T})$ [136], is a procedure that is able to decide the satisfiability of *Boolean combinations* of (quantifier-free) propositional atoms and theory atoms in $\mathcal{T}$.Examples of useful theories are the equality and uninterpreted functions (EUF), difference logic (DL) and linear arithmetic, either over the rationals (LRA) or the integers (LIA), the theories of non-linear real arithmetic (NRA), the theory of bit vectors (BV), and their combinations. The main contributions of Chapter 3 are agnostic of the underlying theory $\mathcal{T}$ [1], while Chapter 4 relies on the NRA and LRA theories.

[1] We require the satisfiability problems for $\mathcal{T}$ to be decidable and to be able to compute Craig interpolants.

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*. Given two formulas ϕ and ψ, with $\phi \wedge \psi \models \bot$, the *Craig Interpolant* (from now on 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. We refer the interested reader to [110] for more details.

2.2 Verification of Symbolic Transition Systems

2.2.1 Symbolic Transition Systems

We represent a discrete-time, infinite-state system with first-order formulas over a background theory $\mathcal{T}$ with signature Σ. Given a set of variables X, we write $X' := \{x' \mid x \in X\}$ for the set of variables copying each variable x with a freshly renamed variable x', and we write $\phi(X)$ if the Σ-formula ϕ only contains free variables from the set X, and abuse the notation writing $\phi(X_1, \ldots, X_n)$ if ϕ contains free variables from $\bigcup_{i=1}^{n} X_i$.

Definition 1 (Symbolic Transition System) *Given a set of state variables X, a Σ-formula $I(X)$ representing a set of initial states, and a Σ-formula $T(X, X')$ is representing a transition relation, $S := \langle X, I, T \rangle$ is a* symbolic transition system.

A *state* s of a transition system S is an interpretation $\langle \mathcal{M}, \mu_s \rangle$ to the symbols in the signature Σ and the variables X, where $\mathcal{M}$ and μ_s are respectively the domain and the assignment of the interpretation. The satisfaction relation $s \models \phi$ for a Σ-formula is defined as usual. We denote with $\mathcal{S}_X$ all the states of the transition system S. We write s' for the state s where the assignments to the variables $x \in X$ from s are substituted with assignments to the variables $x' \in X'$ (i.e., for all $x \in X$, $\mu_s[x] = \mu_{s'}[x']$). A *finite path* (of length k) of S is a finite sequence $\pi := s_0, s_1, \ldots, s_k$ of states with the same domain and interpretation of the symbols in Σ (e.g., for a term $t \in \Sigma$, $\mu_{s_i}[t] = \mu_{s_j}[t]$, for any index i, j) such that $s_0 \models I$, and for all i, $0 \leq i < k$, $s_i, s'_{i+1} \models T$. Observe that the interpretation of the symbols in the signature Σ is *rigid*, meaning that the interpretation to uninterpreted functions and predicates does not change across the states in a path (while the assignments to the variables X can change). We write $\pi \models S$ if π is a path of the transition system S. A state s is *reachable* in S if and only if there exists a path of S ending in s, and we write $\mathcal{S}_S$ for the set of all the reachable states of S.

Example 1 ([J7]) *$S = \langle \{c, d\}, c = 0 \wedge d = 0, c' = c + d \wedge d' = d + 1 \rangle$ is an infinite-state transition system, where $\{c, d\}$ are integer variables. The initial state s_0 of the system has an assignment μ_{s_0} where $\mu_{s_0}[c] = 0$ and $\mu_{s_0}[d] = 0$. At every transition, the system S increases d by one and increases c by d. The path $\pi := s_0, s_1, s_2$ where $s_0 \models c = 0 \wedge d = 0$, $s_1 \models c = 0 \wedge d = 1$, and $s_2 \models c = 1 \wedge d = 2$, is a path of the system S.*

An *infinite path* $\sigma := s_0, s_1, \ldots$ of a transition system S is such that all the states have the same domain and interpretation of symbols in Σ, $s_0 \models I$ and for all $i > 0$ $s_{i-1}, s'_i \models T$.

Example 2 ([J7]) *Consider the transition system $S = \langle \{c, d\}, c = 0 \wedge d = 0, (c' = 0 \wedge d' = 0) \vee (c' = c + d \wedge d' = d + 1) \rangle$. The path that keeps incrementing the value of the c and d variables is an infinite path (i.e., the path with the assignments $\{c = 0, d = 0\}, \{c = 0, d = 1\}, \{c = 1, d = 2\}, \{c = 3, d = 3\}, \ldots$). The path where c and d never change value (i.e., $\{c = 0, d = 0\}, \{c = 0, d = 0\}, \{c = 0, d = 0\}, \ldots$) is also an infinite path of S.*

Given two transition systems $S_1 := \langle X_1, I_1, T_1 \rangle$ and $S_2 := \langle X_2, I_2, T_2 \rangle$, their synchronous product is $S_1 \times S_2 := \langle X_1 \cup X_2, I_1 \wedge I_2, T_1 \wedge T_2, \rangle$. In the following, we assume to represent the transition systems over a theory $\mathcal{T}$ with a signature Σ, so we leave Σ implicit in the notation.

2.2.2 Verification Problems

We are interested in the invariant and liveness verification problems for a symbolic transition system S.

Definition 2 (Invariant Model Checking Problem) *The* invariant verification *problem for a transition system S and a σ-formula $P(X)$ is to decide if all the reachable states of S satisfy P (i.e., for all reachable states $s \in \mathcal{S}_S$ of S, $s \models P$). In such case, we say that the transition system S satisfies the safety property P ($S \models P$).*

The Σ-formula $P(X)$ specifies an invariant property of S, a set of "safe" states. The invariant verification problem is the dual of the *reachability* problem, which asks if there exists a reachable state s such that $s \not\models P$ (observe that the logical negation $\neg P$ represents a set of "unsafe" states).

In this thesis, we will tackle the liveness model checking problem. We briefly recall the *Linear Time Temporal Logic* (LTL) model checking problem [6] and that such problem can be solved via a reduction to a liveness model checking problem. We use the standard syntax of LTL but where the atomic propositions in the formulas are Σ-formulas. A Σ-formula is a LTL formula, and a Boolean combination of LTL formulas is a LTL formula. If ψ_1 and ψ_2 are LTL formulas, then $\mathbf{X}\,\psi_1$ and $\psi_1\,\mathbf{U}\,\psi_2$ are LTL formulas obtained applying the temporal operators next and until, respectively. We derive the other temporal operators finally and globally as usual (i.e., $\mathbf{F}\,\psi := T\,\mathbf{U}\,\psi$, $\mathbf{G}\,\psi := \neg\mathbf{F}\,\neg\psi$). Given an infinite path $\sigma := s_0, s_1, \ldots$, we write σ^i for the suffix of σ starting at state s_i. We define when an infinite path σ satisfies a LTL formula ψ, written $\sigma \models \psi$, by induction:

$$\sigma \models \phi \textbf{ iff } \sigma[0] \models \phi, \quad \sigma \models \neg\psi_1 \textbf{ iff } \sigma[0] \not\models \psi_1, \quad \sigma \models \psi_1 \wedge \psi_2 \textbf{ iff } \sigma[0] \models \psi_1 \text{ and } \sigma[0] \models \psi_2,$$

$$\sigma \models \mathbf{X}\,\psi_1 \textbf{ iff } \sigma^1 \models \psi_1, \qquad \sigma \models \psi_1\,\mathbf{U}\,\psi_2 \textbf{ iff for some } j \geq 0, \sigma^j \models \psi_2 \text{ and for all } i < j, \sigma^i \models \psi_1.$$

Moreover, we assume that the interpretation of all the symbols in the signature Σ in all the states in the path σ is rigid, meaning that the interpretation of function symbols and predicates in σ is the same in the path (while the value assigned to a variable $x \in X$ can change across states in σ). A transition system S satisfies a LTL property ψ ($S \models \psi$) if all the *infinite paths* of S satisfy ψ. The LTL model checking problem consists of checking $S \models \psi$.

Example 3 ([J7]) *Consider the LTL formula $\mathbf{FG}\ c < d$ and the path $\sigma = \{c = 0, d = 0\}, \{c = 0, d = 1\}^\omega$ that repeats $\{c = 0, d = 1\}$ infinitely often. The path satisfies the formula $\mathbf{FG}\ c < d$ because it assigns initially both c and d to 0, and then c to 0 and d to 1 forever.*

Remark 1 *We use the symbol $\models$ with different denotations. If ϕ and ψ are formulas, $\phi \models \psi$ denotes that ψ is a logical consequence of ϕ. If $\langle \mathcal{M}, \mu_s \rangle$ is an interpretation, $\langle \mathcal{M}, \mu_s \rangle \models \psi$ denotes that $\langle \mathcal{M}, \mu_s \rangle$ is a model of ψ. If S is a transition system and ψ an invariant property, $S \models \psi$ denotes that ψ is an invariant of S. If instead ψ is a LTL formula and σ an infinite-path of the transition system S, we use $\sigma \models \psi$ to denote that the path σ satisfies the LTL formula ψ and $S \models \psi$ to denote that all the infinite paths of S satisfy ψ. Different usages of $\models$ will be clear from the context.*

Definition 3 (Liveness Model Checking Problem) *Given a transition system S and a LTL formula $\mathbf{FG}\ \neg f$, with $f(X)$ a Σ-formula, the* liveness model checking problem *decides if $S \models \mathbf{FG}\ \neg f$.*

We adopt the *automata-based approach* [7] to LTL verification that reduces the LTL model checking problem to a liveness model checking problem. The reduction builds a transition system $S_{\neg\psi}$ and a fairness condition $f_{\neg\psi}$ (a Σ-formula) such that:

$$S \models \psi \text{ if and only if } S \times S_{\neg\psi} \models \mathbf{FG}\ \neg f_{\neg\psi}.$$

Thus, in the rest of the thesis we focus on the liveness model checking problem $S \models \mathbf{FG}\ \neg f$.

Remark 2 *Here, we assume the next temporal operator (*$\mathbf{X}$*) to be used only on LTL formulas and not terms of a* Σ*-formula (e.g., the formula* $\mathbf{G}\ ((\mathbf{X}\ x) = x)$ *uses the next operator on the term* x *to expresses that the variable* x *never changes it's initial value). There exists works (e.g., [130]) handling such extensions with a* VMT*-based approach.*

2.3 Verification of Hybrid and Dynamical Systems

We first introduce dynamical and hybrid systems and then the invariant and liveness model checking problems.

In the following, we will refer to the set of formulas that are a Boolean combination of polynomial constraints only containing real-valued variables from a set X as $\Theta_{\text{Poly}}(X)$.

Definition 4 (Dynamical Systems) *A dynamical system* D *is a tuple:*

$$D \ := \ \langle X, \mathit{Init}, \mathit{Inv}, \mathit{Flow} \rangle,$$

where X *is a finite set of real-valued variables,* $\dot{X} := \{\dot{x} \mid x \in X\}$ *is the set of first derivatives of the variables* X*, Init and Inv are formulas from* $\Theta_{Poly}(X)$*, and Flow is a formula in* $\Theta_{Poly}(X \cup \dot{X})$*.*

A state s of the dynamical system D is an interpretation $\langle \mathcal{M}, \mu_s \rangle$ assigning a value to each variable in X. When working with dynamical systems we will assume, without loss of generality, that the set of variables X is $\{x_1, \ldots, x_n\}$, so we can refer to a variable with an index from 1 to n and use a vector notation when more convenient. That's it, sometimes we will work with the vector notation $\vec{X} := [x_1, \ldots, x_n]^T$ instead of the set X. Given a vector $\vec{v}$, we write $\vec{v}_i$ for the i-th element of the vector (e.g., $[x_1, x_2]^T_2$ is x_2). Also, given an assignment μ to the real-valued variables X we write $\vec{\mu} := [\mu[x_1], \ldots, \mu[x_n]]^T$ for the vector of values assigned to the variables X. The dynamical system D reaches a state s' if there exists a state s, a differentiable function $\varphi : \mathbb{R}^{n+1} \to \mathbb{R}^n$, and a non-negative time $t \geq 0$ such that:

1. s is initial (i.e., $s \models \text{Init}$); and
2. $\varphi(\vec{\mu}_{s_0}, 0) = \vec{\mu}_{s_0}$ and $\varphi(s_0, \delta) = \vec{\mu}'_s$; and
3. for all $0 \leq \delta \leq t$:
   - $s_\delta = \langle \mathcal{M}, \bigcup_{x_i \in X} x_i \mapsto \varphi(\vec{\mu}_s, \delta)_i \rangle$ and $s_\delta \models \text{Inv}$, and
   - $s, \langle \mathcal{M}, \bigcup_{x_i \in X} \dot{x}_i \mapsto \left(\frac{d}{dt}\varphi\right)(\vec{\mu}_{s_\delta}, \delta)_i \rangle \models \text{Flow}(X, \dot{X})$, with $\frac{d}{dt}\varphi$ the first derivative with respect to time of φ.

In the above definition, the system dynamic $\text{Flow}(X, \dot{X})$ is intentionally left very general (e.g., Flow may express a polynomial differential algebraic equation). In practice, we often restrict $\text{Flow}(X, \dot{X})$ to a specific subclass to obtain a more amenable verification problem. In this thesis, we will consider the following dynamics:

1. *Piecewise-Constant (PWC) dynamics* (in Section 4.3): Flow is a conjunction of formulas in Linear Real Arithmetic (LRA) of the form: $\sum_{v \in X \cup \dot{X}} (\alpha_v \cdot v) \bowtie 0$, where each coefficient $\alpha_v \in \mathbb{Q}$.
2. *Polynomials dynamics* (in Sections 4.1 and 4.2): Flow expresses a system of Ordinary Differential Equations (ODEs) of the form $\dot{\vec{X}} = \vec{f}(\vec{X})$, where $\dot{\vec{X}}$ is the vector of first-order derivatives of the variables $\vec{X}$, $\vec{f}(\vec{X})$ is a vector of polynomials using the variables X (i.e., each polynomial $\vec{f}_i$ describes the right-hand side of the differential equation for the variable x_i, $\dot{x}_i = \vec{f}_i(\vec{X})$).

Remark 3 *In order to ease the presentation we restricted the Flow condition to polynomials with variables from X (i.e., $\Theta_{Poly}(X)$). We observe that some of the techniques we propose in Chapter 4 can handle more expressive dynamical systems. For example, the above definition does not include in the vector field parameters (which are supported in the algorithm of Section 4.1), inputs variables, and disturbances (which are supported in Section 4.2). Moreover, the vector field is polynomial, while the algorithm from Section 4.2 considers continuous dynamics with transcendental functions.*

Definition 5 (Hybrid Automata) *A hybrid automata [8, 113] is a finite automaton extended with a set of continuous variables X defined as:*

$$H := \langle Q, R, X, \mathit{Init}, \mathit{Inv}, \mathit{Jump}, \mathit{Flow} \rangle,$$

where Q is a finite set of states, X is a finite set of real-valued variables, $\mathit{Init} : Q \to \Theta_{Poly}(X)$, $\mathit{Inv} : Q \to \Theta_{Poly}(X)$, $\mathit{Jump} \subseteq Q \times \Theta_{Poly}(X, X') \times Q'$ is a transition relation, and $\mathit{Flow} : Q \to \Theta_{Poly}(X, \dot{X})$ is a flow condition imposing a relation among the derivatives $\dot{X}$ and the continuous variables X.

A state (q_i, s_i) transitions to a state (q_{i+1}, s_{i+1}) if either:

- *Discrete step*: $s_i \models \mathsf{Inv}(q_i)$, $s_{i+1} \models \mathsf{Inv}(q_{i+1})$, $(q_i, \phi, q_{i+1}) \in \mathsf{Jump}$ and $s_i, s_{i+1} \models \phi$; or
- *Time elapse*: $q_i = q_{i+1}$, $s_i \models \mathsf{Inv}(q_i)$, $s_{i+1} \models \mathsf{Inv}(q_i)$, s_i reaches s_{i+1} in the dynamical system $\langle X, \bigwedge_{x \in X} \dot{x} = \mu_{s_i}[x], \mathsf{Inv}(q_i), \mathsf{Flow}(q_i) \rangle$.

A finite path $\pi := (q_0, s_0), \ldots, (q_k, s_k)$ of H is a sequence of states such that $s_0 \models \mathsf{Init}(q_0)$ and (q_i, s_i) transitions to (q_{i+1}, s_{i+1}) for $i < k$. We define an infinite path σ analogously, but for an infinite sequence of states. Let δ_i be the time elapsed during the time elapse transition from (q_i, s_i) to (q_{i+1}, s_{i+1}). The total time t_i elapsed in total at the state (q_i, s_i) on a path π is 0 if $i = 0$, t_{i-1} if the transition from (q_i, s_i) to (q_{i+1}, s_{i+1}) was a discrete step, and $t_i = t_{i-1} + \delta_{i-1}$ if the transition (q_i, s_i) to (q_{i+1}, s_{i+1}) was a continuous step. An infinite path σ is *zeno* if the sequence of times $t_o, t_1, \ldots$ does not diverge.

We define the invariant model checking problem for a hybrid systems H and an invariant property P (written as $H \models P$) similarly to Definition 2. In practice, the invariant model checking problem asks to prove that all the finite paths of the hybrid system H never reaches a state outside $P(X)$ (here, we also assume that we can express automata locations Q in the invariant formula P, which is feasible encoding the set Q with a set of Boolean variables). We define the liveness verification problem analogously to Definition 3, and say that $H \models \mathbf{FG}\, \neg f$ if all the infinite paths σ of H are such that $\sigma \models \mathbf{FG}\, \neg f$.

Chapter 3

Verification Modulo Theories

This chapter presents the work from the following papers: [C12, J5, C17, J7].

There is a well established set of algorithms for model checking symbolic transition systems with a finite number of states that are commonly used to verify hardware and high-level system models. Among such model checking algorithms, the ones that use a SAT solver to reason symbolically on the transition system (e.g., Bounded Model Checking (BMC) [12], K-induction [17], K-liveness [65], liveness to safety (L2S) [20], interpolation-based model checking [25], IC3 [55], ...) demonstrated to scale to verify properties on complex systems. Such algorithms are appealing since they search for an inductive invariant sufficient to prove a property, instead of eagerly computing the set of reachable states like the algorithms based on Binary Decision Diagrams. Extending the SAT-based model checking algorithms to verify infinite-state transition systems expressed with theories poses several challenges. The main operation the above algorithms perform is to query a SAT solver checking the satisfiability of a propositional logic formula repeatedly (and in some cases computing other facts, such as Craig interpolants). Since the SMT problem subsumes the SAT problem, a naïve idea to extend SAT-based algorithms to model check infinite-state systems is to just replace the SAT solver with a SMT solver. However, such strategy does not work well when we have an infinite number of states since (i) when proving an invariant property such algorithms end up enumerating an infinite amount of states (i.e., they do not converge); and (ii) when proving liveness properties such algorithms fail since they assume the system has only lasso-shaped paths (i.e., a path formed by a prefix followed by a "loop", where one state in the path is visited infinitely often), while paths in an infinite-state system may not have a "loop". In this Chapter, we answer the following question:

How can we *effectively* use SMT-based model checking algorithms to model check infinite-state transition systems?

We first focus on the problem of extending the IC3 algorithm to prove invariant properties on infinite-state systems (Section 3.1) and then on problem of model checking problems liveness properties (Section 3.2). All the algorithm we present in this Chapter have been implemented in the *nuXmv* [C11] symbolic model checker.

3.1 IC3-IA: Extending IC3 with Implicit Predicate Abstraction

In this Section, we first provide a high-level overview of the IC3 model checking algorithm, focusing on the challenges to extend it to the theory case, then we recall the CEGAR loop for

predicate abstraction, and finally we present IC3ia.

3.1.1 SAT-based IC3.

The IC3 algorithm [55], or Property Directed Reachability (PDR) algorithm [57], tries to incrementally build an inductive invariant to prove $S \models P$. An inductive invariant is a formula ψ such that: (i) $\psi \models P$; and (ii) $I \models \psi$; and (iii) $\psi \wedge T \models \psi'$ that is a sufficient certificate to conclude that $S \models P$.Observe that the set of reachable states $\mathcal{S}_S$ of S is the strongest inductive invariant (i.e., for any inductive invariant ψ we have $\mathcal{S}_S \models \psi$).

IC3 maintains a finite sequence of frames $\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_k$ called *trace*, where each *frame* $\mathcal{F}_i$ is an over-approximation of the set of states S reaches after i steps.[1] Moreover, the trace satisfies the following properties:

$$\mathcal{F}_0 := I, \tag{3.1}$$

$$\mathcal{F}_i \models \mathcal{F}_{i+1}, \text{ for } 0 \leq i < k, \tag{3.2}$$

$$\mathcal{F}_i \wedge T \models \mathcal{F}'_{i+1}, \text{ for } 0 \leq i < k, \tag{3.3}$$

$$\mathcal{F}_i \models P, \text{ for } 0 \leq i < k. \tag{3.4}$$

An important aspect in the mechanics of the IC3 algorithm is that each frame $\mathcal{F}$ is represented with a set of clauses and not an arbitrary Boolean formula. We recall that a clause c is a disjunction of literals (i.e., a Boolean atom or its negation) $l_1 \vee \cdots \vee l_n$ and that we can write c as a set $\{l_1, \ldots, l_n\}$.

Initially, IC3 checks if $I \models P$ and then initializes the frame $\mathcal{F}_0 = I$. Observe how the current trace, only containing $\mathcal{F}_0$, satisfies the trace properties. At each major iteration, the algorithm tries to extend the current trace with a new frame that initially is true (e.g., at step 1, $\mathcal{F}_1 = \top$). Suppose we have a trace $\mathcal{F}_0, \ldots, \mathcal{F}_{k-1}$ satisfying the trace properties and IC3 extends the trace with the new frame $\mathcal{F}_k = \top$. The new trace including $\mathcal{F}_k$ satisfies the trace properties and $\mathcal{F}_k \not\models P$. Thus, there is at least a state $s_k \models \mathcal{F}_k$ and $s_k \not\models P$.

IC3 performs a *blocking phase* to either prove that s_k cannot be reached after k steps, or that there is a path from the initial states I reaching s_k. The blocking phase first check if the previous frame, $\mathcal{F}_{k-1}$, can block the pair (s_k, k) (also called proof obligation), checking if $\neg s_k \wedge \mathcal{F}_{k-1} \wedge T \models \neg s'_k$. In practice, if such check succeeds we have that, when taking a transition from a state in $\neg s_k \wedge \mathcal{F}_{k-1}$, we always reach a state in $\neg s_k$, hence proving that s_k is not reachable in k steps. Such condition is called *relative induction*:

$$RelInd(\mathcal{F}, T, c) := c \wedge \mathcal{F} \wedge T \wedge \neg c', \tag{3.5}$$

stating that the clause c is inductive relative to the frame $\mathcal{F}$. If the formula $RelInd(\mathcal{F}_{k+1}, T, \neg s_k)$ is unsatisfiable, then the IC3 inferred that s_k cannot be reached in k steps. Thus, IC3 refines $\mathcal{F}_k$ as $\mathcal{F}_k \wedge \neg s_k$ (note that $\neg s_k$ is a clause). At this point, IC3 checks again if $\mathcal{F}_k \models P$, eventually performing the same blocking phase. Instead, when $RelInd(\mathcal{F}_{k+1}, T, \neg s_k)$ fails in blocking s_k, then there is a state s_{k-1} in $\mathcal{F}_{k-1}$ that can reach s in $\mathcal{F}_k$. The blocking phase recursively tries to block $(s_{k-1}, k-1)$ with the frame $\mathcal{F}_{k-2}$. In practice, such recursive check will eventually terminate, either succeeding in proving that $\mathcal{F}_k \not\models s_k$ (i.e., "blocking" s_k), or finding a path starting in $\mathcal{F}_0$ reaching s_k (i.e., a counterexample path showing $S \not\models P$).

IC3 executes the *propagation phase* after it proves that $\mathcal{F}_k \models P$. In the propagation phase, IC3 check if a clause $c \in \mathcal{F}_i$ is relative inductive (i.e., $RelInd(\mathcal{F}_i, T, c)$). If that is the case, then c can be added (propagated) to the following frame $\mathcal{F}_{i+1}$, strengthening it. The propagation

[1] In this section, we use the term trace to refer to the sequence of frames in the IC3 algorithm, and not to refer to a trace of a transition system.

processes all the frames in order, starting from the frame $\mathcal{F}_0$. The propagation phase allows IC3 to detect a fixed point in the algorithm when $\mathcal{F}_{k-1} = \mathcal{F}_k$. Using the trace properties it's not difficult to see that $\mathcal{F}_{k-1}$ is an inductive invariant when $\mathcal{F}_{k-1} = \mathcal{F}_k$ (i.e., $I \models \mathcal{F}_{k-1}$, $\mathcal{F}_{k-1} \models P$, and $\mathcal{F}_{k-1} \wedge T \models \mathcal{F}_{k-1}$).

The above description abstracts several optimizations that are key for IC3 performance and, in particular, the *generalization* of the inductive clauses. After the algorithm finds that the clause c is relative inductive to $\mathcal{F}$ in the blocking phase, i.e., $RelInd(\mathcal{F}, T, c)$ is unsatisfiable, it tries to generalize the clause c to a weaker clause g (i.e., a clause g such that $c \models g$), which potentially blocks more states. The generalization procedures for IC3 are described in [57, 100].

Challenges in Extending IC3 to SMT. The above high-level description of the IC3 algorithm is agnostic of the underlying type of system (e.g., finite- or infinite-state) and the fact that formulas are purely propositional or Σ–formulas. The extension to the SMT case presents some challenges, as explained in [64, 67]. In the finite-state case, when the *RelInd* check fails in the blocking phase for a proof obligation (s_k, k), one get a state s_{k-1} from the previous frame $\mathcal{F}_{k-1}$ from a satisfying assignment μ to the formula $\neg s_k \wedge \mathcal{F}_k \wedge T \wedge s'_k$. In fact, one can get s_{k-1} from the assignment μ just considering the assignment to the "current" variables (i.e., ignoring the "primed" variables). Note that, given the structure of the frames, s_{k-1} must be a cube (i.e., a conjunction of literals). In the SMT settings, one can get a "trivial" cube s_{k-1} from the model μ, assigning to each variable of the transition system a single value. Such cube s_{k-1} will have the effect to eliminate a single state from $\mathcal{F}_{k-1}$ when the state space is infinite.

There are two main orthogonal solutions to extend IC3 to the theory case:

1. Obtain a "larger" cube s_{k-1} that represents a set (instead of one) of predecessors of s_k. We can compute generalization computing the pre-image of s_k using quantifier elimination [64]. In practice, since quantifier elimination is expensive, one can use an under-approximate version of quantifier elimination to obtain a subset of all the predecessor states (e.g., see model-based quantifier elimination [50, 151]);

2. Extend the generalization step in the blocking phase [64] to obtain a "larger" blocking clause. We can compute such generalization using theory specific interpolation (e.g., see the use of Farka's Lemma in [67]).

A disadvantages of the above solutions is that both quantifier elimination and generalization steps are *theory dependent*, requiring to develop an ad-hoc technique for every theory of interest.

3.1.2 CEGAR and Implicit-Predicate Abstraction

An approach for verifying infinite-state systems is *Predicate Abstraction* [9]. In abstract model checking [5], we analyze an abstract transition system $\widehat{S}$, expressed with the variables $\widehat{X}$ (note that $\widehat{S}$ is finite-state if all the variables $\widehat{X}$ have a finite domain) that we obtain with an *abstraction function* $\alpha : \mathcal{S}_X \to \mathcal{S}_{\widehat{X}}$, a surjective function mapping each state of the transition system S into states of $\widehat{S}$. A *concretization function* $\gamma : \mathcal{S}_{\widehat{X}} \to 2^{\mathcal{S}_x}$ concretizes an abstract state $\widehat{s}$ to a set of concrete states, i.e., $\gamma(\widehat{s}) := \{s \in \mathcal{S}_X \mid \alpha(s) = \widehat{s}\}$. We represent the abstraction function α with a formula $H_\alpha(X, \widehat{X})$ such that $s, \widehat{s} \models H_\alpha(X, \widehat{X})$ iff $\alpha(s) = \widehat{s}$. The abstract transition system $\alpha(S) := \langle \widehat{X}, \widehat{I}, \widehat{T} \rangle$, also written as $\widehat{S}$, and the abstract invariant property are

defined as:

$$\widehat{I} := \exists X.(I(X) \wedge H_\alpha(X, \widehat{X})), \tag{3.6}$$

$$\widehat{T} := \exists X, X'.(T(X, X') \wedge H_\alpha(X, \widehat{X}) \wedge H_\alpha(X', \widehat{X'})), \tag{3.7}$$

$$\widehat{P} := \exists X.P(X) \wedge H_\alpha(X, \widehat{X}). \tag{3.8}$$

The abstraction function guarantees that if $\widehat{S} \models \widehat{P}$, then $S \models P$. Hence, the goal of abstract model checking is to construct a verification problem $\widehat{S} \models \widehat{P}$ that is "easier" to solve than $S \models P$. Commonly, such condition is ensured by considering finite-state abstractions.

A widely used finite-state abstraction is *Predicate Abstraction* [9], where we assume a finite set of predicates $\mathbb{P} := \{p_1, \ldots, p_m\}$ with each $p \in \mathbb{P}$ using only variables from X, and a finite set of Boolean variables $\widehat{X} := \{x_p \mid p \in \mathbb{P}\}$ (we will write $X_\mathbb{P}$ for $\widehat{X}$ when having a predicate abstraction). We represent the abstraction function $\alpha_\mathbb{P}$ as: [2]

$$H_\mathbb{P}(X, X_\mathbb{P}) := \bigwedge_{p \in \mathbb{P}} p(X) \leftrightarrow x_p. \tag{3.9}$$

Intuitively, the predicate abstraction $H_\mathbb{P}$ groups together states that have the same valuation to all the predicates in $\mathbb{P}$.

The main challenges in abstract model checking using predicate abstraction are 1. computing the abstraction, which involves an exponential number of states; and 2. finding a *sufficient* set of predicates to prove the property (when the invariant property holds). The CEGAR framework [24] tackles the former problem, while Implicit Predicate Abstraction [44] tackles the latter one.

Counter-Example Guided Abstraction Refinement (CEGAR). Given an invariant verification problem $S \models P$, we don't know the verification result and, in case $S \models P$, if there exists a finite set of predicates $\mathbb{P}$ such that $\widehat{S} \models \widehat{P}$. In practice, the verification algorithms implement the *Counter-Example Guided Abstraction Refinement* (CEGAR) framework [24] shown in Figure 3.1 that iteratively: [3] (i) computes the abstract system and property $\widehat{S}$ and $\widehat{P}$; (ii) model check the abstract system ($\widehat{S} \models \widehat{P}$), either proving that $S \models P$ or producing an abstract counterexample $\widehat{\pi}$ (i.e., a path in $\widehat{S}$ such that $\widehat{\pi} \not\models \widehat{P}$); (iii) check if there is a concretization of the abstract counterexample $\widehat{\pi}$ that corresponds to a path in S, concluding that $S \not\models P$; (iv) in case $\widehat{\pi}$ is spurious, the algorithm refines the abstraction that, in the context of predicate abstraction, means finding new predicates $\mathbb{P}'$ that rule out $\widehat{\pi}$ in the new abstract system (i.e., $\widehat{\pi} \not\models \alpha_{\mathbb{P}\cup\mathbb{P}'}(S)$). The CEGAR loop then restarts with the new set of predicates $\mathbb{P} \cup \mathbb{P}'$.The simulation and refinement steps when using predicate abstraction are usually based on Bounded Model Checking (BMC) [12]. The concrete system S simulates a spurious counterexample $\widehat{\pi}$ if the following BMC encoding is satisfiable:

$$I(X^0) \wedge \bigwedge_{0 \le i < k} T(X^i, X^{i+1}) \wedge \bigwedge_{0 \le i < k} \left(H_\mathbb{P}(X^i, X^i_\mathbb{P}) \wedge \widehat{\pi}_i(X^i_\mathbb{P})\right), \tag{3.10}$$

where $X^i := \{x^i \mid x \in X\}$ is a set of copies of the set of variables X indexed with $i \in \mathbb{N}$ (and similarly for $\widehat{X}$). The formula is a BMC problem encoding all the paths of S of length k that visit the same sequence of abstract states from $\widehat{\pi}$. If Equation (3.10) is satisfiable, then an interpretation of the variables X represents a concrete counterexample witnessing

[2] Abusing the notation, here we identify the abstraction function with the set of predicates $\mathbb{P}$.

[3] We instantiate CEGAR to predicate abstraction even if CEGAR works with generic abstraction functions α.

the violation of the invariant property P. Otherwise, the counterexample $\widehat{\pi}$ is spurious and the abstraction must be refined. A popular algorithm [27] uses interpolation to find new predicates $\mathbb{P}'$ that will rule out $\widehat{\pi}$ in the new abstract system using $\mathbb{P} \cup \mathbb{P}'$. The procedure computes a sequence of interpolants that is, intuitively, a Craig interpolant between every pair of transitions in the encoding of Equation (3.10). Then, the new predicates $\mathbb{P}'$ are all the predicates contained in the sequence interpolants.

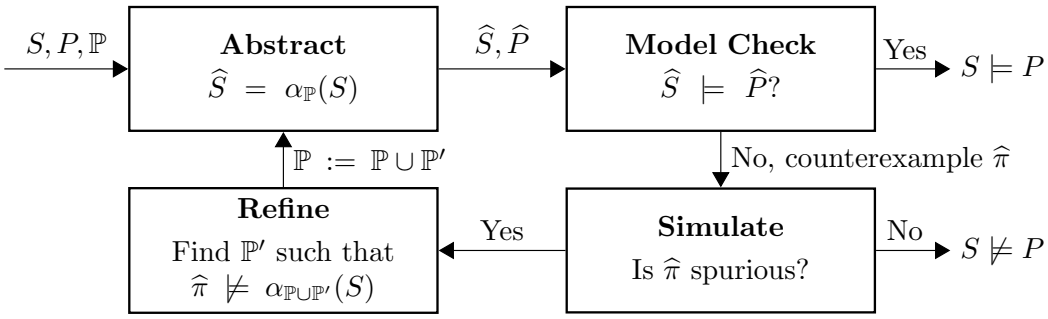


Figure 3.1: CEGAR. Counter-Example Abstraction Refinement loop.

Implicit Predicate Abstraction. One of the challenges when instantiating the CEGAR framework with predicate abstraction is the intrinsic complexity in computing the abstract system $\widehat{S}$. In fact, computing $\widehat{S}$ requires to eliminate the quantifier from the Equation (3.6). While such computation is not expensive for some abstraction (e.g., Cartesian Abstraction [22]), they have exponential complexity in the number of predicates in the case of predicate abstraction. Despite several algorithm tried to address such problem (e.g., [33]), computing the predicate abstraction fundamentally consists of enumerating the truth assignments to the Boolean predicates, which are exponential in the number of predicates. Furthermore, such computation has to be done for all the formulas in Equation (3.6) at every iteration of the CEGAR loop, becoming more expensive as the abstraction becomes more precise. Such computation bottleneck hinders the practical use of CEGAR to verify infinite-state symbolic transition systems and will reappear when proving invariant property for dynamical system later in Section 4.1.

Implicit Predicate Abstraction (IA) [44] uses the intuition that an abstraction α induces an equivalence relation $\sim$ among states such that:

$$s_1 \sim s_2 \text{ iff } \alpha(s_1) = \alpha(s_2).$$

We can express such equivalence relation when the abstraction α is a predicate abstraction with predicates $\mathbb{P}$ as:

$$EQ_{\mathbb{P}}(X, \overline{X}) := \bigwedge_{p \in \mathbb{P}} p(X) \leftrightarrow p(\overline{X}). \tag{3.11}$$

The Equation (3.11) relates two sets of concrete states and is such that $\alpha(s_1) = \alpha(s_2)$ iff $X, \overline{X} \models EQ_{\mathbb{P}}(X, \overline{X})$. Instead of computing the abstraction eagerly, we encode the abstract transition in the symbolic encoding of a path:

$$Path^k_{\mathbb{P}} := \bigwedge_{1 \le i < k} (T(\overline{X}^{i-1}, X^i) \wedge EQ_{\mathbb{P}}(X^i, \overline{X}^i)) \wedge T(\overline{X}^{k-1}, X^k). \tag{3.12}$$

The formula $Path^k_{\mathbb{P}}$ is satisfiable iff there exists an uninitialized path of length k in the *abstract* transition system $\widehat{S}$. [4]

3.1.3 The IC3IA algorithm

The IC3IA algorithm tackles the challenges described above restricting IC3 to reason on the abstract system $\widehat{S}$ so that all the IC3 frames are clauses with literals from the abstract variables $X_{\mathbb{P}}$, and then use implicit abstraction for checking relative inductiveness (Equation (3.5)) to avoid the explicit computation of the abstract transition relation $\widehat{T}$. We encode the abstract relative induction *AbsRelInd* as follows:

$$\begin{aligned} AbsRelInd(\mathcal{F}, T, c, \mathbb{P}) \; := \mathcal{F}(X_{\mathbb{P}}) \wedge c(X_{\mathbb{P}}) \wedge H_{\mathbb{P}}(X, X_{\mathbb{P}}) \wedge H_{\mathbb{P}}(X', X'_{\mathbb{P}}) \wedge \\ EQ_{\mathbb{P}}(X, \overline{X}) \wedge T(\overline{X}, \overline{X}') \wedge EQ_{\mathbb{P}}(\overline{X}', X') \wedge \neg c(X'_{\mathbb{P}}). \end{aligned} \tag{3.13}$$

$AbsRelInd(\mathcal{F}, T, c, \mathbb{P})$ is unsatisfiable iff $RelInd(\mathcal{F}, \widehat{T}, c)$ is unsatisfiable (see [J5] for a proof).

The IC3IA algorithm further implements a CEGAR loop to automatically check if an abstract counterexample $\widehat{\pi}$ is spurious and, in that case, refine the set of predicates $\mathbb{P}$. [5] Observe that, the frames IC3 computes in a CEGAR iteration can be reused in the subsequent iterations when considering the refined set of predicates $\mathbb{P} \cup \mathbb{P}'$. In fact, the invariant on the IC3 frames from Equations (3.1), (3.2), (3.3), and (3.4) will also hold with the new abstract transition computed with the predicates $\mathbb{P} \cup \mathbb{P}'$ (intuitively, the new abstract transition implies the old one).

The IC3IA algorithm is sound (i.e., when the algorithm is correct when terminating, concluding either that $S \models P$ or $S \not\models P$) and the refinement guarantees progress [J5]. Similarly to what happens with other CEGAR algorithms based on predicate abstraction, there is no guarantee IC3IA will terminate in case $S \models P$. In fact, even when a sufficient set of predicates $\mathbb{P}$ exists, the algorithm may perform an infinite number of CEGAR loops (e.g., see [32]). In practice, the algorithm terminates for finite domains (e.g., the Bit-Vector theory), and may terminate adapting ad-hoc refinement strategies for particular classes of systems (e.g., for timed automata [68] the refinement may add as predicates clock constraints describing the region graph).

Understanding IC3IA (Example from [J5]). To give an intuition of the IC3IA behavior, we show its main steps when proving the property $P := (d \leq 3) \vee \neg(c \leq d)$ for the transition system $S = \langle \{c, d\}, c = 0 \wedge d = 0, c' = c + d \wedge d' = d + 1 \rangle$. We report the full example as presented in [J5]. We describe the first three iterations of the IC3IA main loop, showing all the important steps of the algorithm, like the blocking phase using the abstract relative induction check and the refinement.

IC3IA proves that the property holds performing 8 iterations of the IC3IA main loop, refining 4 times the abstraction and ending with a total of 9 predicates. The initial set of predicates, taken from the initial formula and the property, is $\mathbb{P}_0 := \{(c = 0), (d = 0), (d \leq 3), (c \leq d)\}$ and the initial status of the frames is $\mathcal{F}_0 := x_{c=0} \wedge x_{d=0}$.

(I) **First iteration.** In the first iteration the algorithm checks that $\mathcal{F}_0 \wedge H_{\mathbb{P}}(X, X_{\mathbb{P}}) \wedge \neg P$ is unsatisfiable, adding the empty frame $\mathcal{F}_1$.

(II) **Second iteration.** IC3IA finds a pair $(c_0, 1)$ where $c_0 = x_{c=0} \neg x_{d=0} \wedge \neg x_{d \leq 3} \wedge x_{c \leq d}$ and such that $c_0 \wedge H_{\mathbb{P}}(X, X_{\mathbb{P}}) \models \mathcal{F}_1 \wedge \neg P$. Then, IC3IA tries to block $(c_0, 1)$ in the

[4] We'll see in Section 4.1 that we can construct a symbolic transition system encoding directly $\widehat{S}$ using IA.

[5] The architecture is similar to the one in Figure 3.1, with the abstraction and model checking phases merged together.

frame $\mathcal{F}_0$: c_0 is blocked by $\mathcal{F}_0$, since $AbsRelInd(\mathcal{F}_0, T, c_0, \mathbb{P}_0)$ is unsatisfiable. In fact, $AbsRelInd(\mathcal{F}_0, T, c_0, \mathbb{P}_0)$ is the formula:

$$
\begin{aligned}
AbsRelInd(\mathcal{F}_0, T, c_0, \mathbb{P}_0) \;:=\; & x_{c=0} \wedge x_{d=0} \wedge && [\mathcal{F}_0(X_{\mathbb{P}})]\\
& x_{c=0} \wedge \neg x_{d=0} \wedge \neg x_{d\leq 3} \wedge x_{c\leq d} \wedge && [c_0(X_{\mathbb{P}})]\\
& x_{d=0} \leftrightarrow (d=0) \wedge x_{c=0} \leftrightarrow (c=0) \wedge && [H_{\mathbb{P}}(X, X_{\mathbb{P}})]\\
& x_{d\leq 3} \leftrightarrow (d \leq 3) \wedge x_{c\leq d} \leftrightarrow (c \leq d) \wedge &&\\
& x'_{d=0} \leftrightarrow (d'=0) \wedge x'_{c=0} \leftrightarrow (c'=0) \wedge && [H_{\mathbb{P}}(X', X'_{\mathbb{P}})]\\
& x'_{d\leq 3} \leftrightarrow (d' \leq 3) \wedge x'_{c\leq d} \leftrightarrow (c' \leq d') \wedge &&\\
& (d=0) \leftrightarrow (\overline{d}=0) \wedge (c=0) \leftrightarrow (\overline{c}=0) \wedge && [EQ_{\mathbb{P}}(X, \overline{X})]\\
& (d \leq 3) \leftrightarrow (\overline{d} \leq 3) \wedge (c \leq d) \leftrightarrow (\overline{c} \leq \overline{d}) \wedge &&\\
& (\overline{c'} = \overline{c} + \overline{d}) \wedge (\overline{d'} = \overline{d} + 1) \wedge && [T(\overline{X}, \overline{X}')]\\
& (\overline{d'}=0) \leftrightarrow (d'=0) \wedge (\overline{c'}=0) \leftrightarrow (c'=0) \wedge &&\\
& (\overline{d'} \leq 3) \leftrightarrow (d' \leq 3) \wedge (\overline{c'} \leq \overline{d'}) \leftrightarrow (c' \leq d') \wedge && [EQ_{\mathbb{P}}(\overline{X}', X')]\\
& \neg(x'_{c=0} \wedge \neg x'_{d=0} \wedge \neg x'_{d\leq 3} \wedge x'_{c\leq d}). && [\neg c_0(X'_{\mathbb{P}})]
\end{aligned}
$$

Since $AbsRelInd(\mathcal{F}_0, T, c_0, \mathbb{P}_0) \models \bot$, IC3IA tries to generalize c_0 to block more states in the frame $\mathcal{F}_1$. One possible generalization is $\neg x_{d\leq 3}$, since $AbsRelInd(\mathcal{F}_0, T, \neg x_{d\leq 3}, \mathbb{P}_0) \models \bot$. IC3IA adds the negation of the generalized cube, $x_{d\leq 3}$, to $\mathcal{F}_1$. Now the frame $\mathcal{F}_1$ does not intersect the bad states $\neg P$, and thus IC3IA adds the frame $\mathcal{F}_2$ and proceeds to the propagation phase (in this case there are no clauses in a frame that can be propagated to the successive frame).

(III) **Third iteration.** IC3IA finds a chain of pairs: $(\neg x_{c=0} \wedge \neg x_{d=0} \wedge \neg x_{d\leq 3} \wedge x_{c\leq d}, 2)$, $(x_{c=0} \wedge \neg x_{d=0} \wedge x_{d\leq 3} \wedge x_{c\leq d}, 1)$ and $(x_{c=0} \wedge x_{d=0} \wedge x_{d\leq 3} \wedge x_{c\leq d}, 0)$ by finding a satisfiable assignment to $\neg x_{c=0} \wedge \neg x_{d=0} \wedge \neg x_{d\leq 3} \wedge x_{c\leq d} \wedge H_{\mathbb{P}}(X, X_{\mathbb{P}}) \models \mathcal{F}_2 \wedge \neg P$, and then recursively calling the blocking function. The last pair is at depth 0, hence IC3IA found an abstract counterexample. The counterexample path cannot be simulated on the concrete system, due to the transition from the second to the third state of the path. In the third state we have that the abstract path requires that $\neg(d \leq 3)$, but in the concrete system d must be lower or equal than 2 after two steps. The refinmement finds $(d \leq 2)$ as new predicate; now the abstraction is determined by the set of predicates $\mathbb{P}_1 := \mathbb{P}_0 \cup \{(d \leq 2)\}$.

After the refinement, IC3IA checks if there exists another cube that violates P at frame $\mathcal{F}_2$. The search still finds the pairs: $(\neg x_{c=0} \wedge \neg x_{d=0} \wedge \neg x_{d\leq 3} \wedge x_{c\leq d}, 2)$ and $(x_{c=0} \wedge \neg x_{d=0} \wedge x_{d\leq 3} \wedge x_{c\leq d} \wedge \neg x_{d\leq 2}, 1)$. $(x_{c=0} \wedge \neg x_{d=0} \wedge x_{d\leq 3} \wedge x_{c\leq d} \wedge \neg x_{d\leq 2}, 1)$ is blocked by $\mathcal{F}_0$, and thus IC3IA adds $x_{d\leq 2}$ to $\mathcal{F}_1$; then $(\neg x_{c=0} \wedge \neg x_{d=0} \wedge \neg x_{d\leq 3} \wedge x_{c\leq d}, 2)$ is blocked by $\mathcal{F}_1$, thus IC3IA adds $x_{d\leq 3}$ to $\mathcal{F}_2$. At this point $\mathcal{F}_2$ satisfies the property and IC3IA adds the frame $\mathcal{F}_3$, peforming the propagation phase (it still does not propagate any clause).

(IV) **Final result.** The final set of predicates found by IC3IA is $\{(c=0), (d=0), (d \leq 3), (c \leq d), (d \leq 2), (d \leq 1), (1 \leq c), (3 \leq c)\}$ and the final inductive invariant is:

$$
\begin{aligned}
&(\neg(c=0) \vee (d \leq 2)) \wedge ((d \leq 1) \vee (1 \leq c)) \wedge ((c=0) \vee \neg(d \leq 1)) \wedge\\
&(\neg(c=0) \vee (c \leq d)) \wedge ((d \leq 2) \vee (1 \leq c)) \wedge ((d \leq 2) \vee (3 \leq c)) \wedge ((d \leq 3) \vee \neg(c \leq d)).
\end{aligned}
$$

3.2 L2S−IA: Liveness-to-Safety Reductions via Implicit Abstraction

Several liveness verification algorithms work under the assumptions that all the infinite paths of a system are *lasso-shaped*. An infinite path is lasso shaped if it can be divided in a finite length prefix and a loop repeating infinitely often. While such assumption holds for finite-state systems, it is not the case for infinite-state systems. Here, we focus on the Liveness-to-Safety (L2S) [20] reduction algorithm for liveness verification. In the L2S algorithm, a *finite-state* transition system $S \models \mathbf{FG}\ \neg f$ if and only if the transition system S_{L2S} is such that $S_{\text{L2S}} \models \neg\text{loop}$, where the transition system $S_{\text{L2S}} := \{X_{\text{L2S}}, I_{\text{L2S}}, T_{\text{L2S}}\}$ is:

$$X_{\text{L2S}} := X \cup \overline{X} \cup \{\text{seen}, \text{triggered}, \text{loop}\}, \tag{3.14}$$

$$I_{\text{L2S}} := I \wedge \neg\text{seen} \wedge \neg\text{triggered} \wedge \neg\text{loop}, \tag{3.15}$$

$$T_{\text{L2S}} := T \wedge \bigwedge_{x \in X} \overline{x} = \overline{x}' \wedge \tag{3.16}$$

$$\left(seen' \leftrightarrow \left(seen \vee \bigwedge_{x \in X} x = \overline{x}\right)\right) \wedge \tag{3.17}$$

$$(triggered' \leftrightarrow (triggered \vee (f \wedge seen'))) \wedge \tag{3.18}$$

$$\left(loop' \leftrightarrow \left(triggered' \wedge \bigwedge_{x \in X} x' = \overline{x}'\right)\right). \tag{3.19}$$

The intuition of the above encoding is to:

1. non-deterministically guess the start of the loop of a lasso path, setting the variable seen to true and "storing" the current value of the state variables to $\overline{X}$; and
2. setting the triggered variable to true when the fairness f holds in a state *inside* the loop; and
3. setting the loop variable to true when seeing a repetition of the loop (i.e., when $\bigwedge_{x \in X} x = \overline{x}$ after the loop start).

What is appealing about the L2S reduction is that the encoding transforms a fairness verification problem to an invariant verification problem (e.g., checking if $S_{\text{L2S}} \models \neg\text{loop}$), and allows us to use any SAT-based verification algorithm. However, the L2S reduction cannot prove that $S \models \neg\text{loop}$ when S is an infinite-state system, since $S_{\text{L2S}} \models \neg\text{loop}$ does not imply $S \not\models \mathbf{FG}\ \neg f$.

Example 4 ([J7]) *To see the limitation of L2S, consider the infinite-state transition system $S = \langle \{c, d\}, c \geq 0, (c' = c + d \wedge d' = d + 1) \rangle$, with c and d integers, and the verification problem $S \models \mathbf{FG}\ c < d$. We have that $S \not\models \mathbf{FG}\ c < d$: independently from the non-deterministic initial value of d, d will eventually be positive and hence, eventually, c will be greater than d. However, the transition system S has only infinite paths that are **not** lasso-shaped, and that are not "considered" in the S_{L2S} transition system (i.e., in the transition system S_{L2S} the variable for seeing a loop will never be true). Thus, in this example we have that $S_{L2S} \models \mathbf{FG}\ \neg f$ and write $S \not\models \mathbf{FG}\ \neg f$.*

In [C17] we show that the L2S reduction is useful when applied to the predicate abstraction of an infinite-state transition system S. Clearly, the abstract system $\widehat{S}$ is finite state, so $\widehat{S}_{\text{L2S}} \models \mathbf{FG}\ \neg\widehat{f}$ implies $\widehat{S} \models \mathbf{FG}\ \neg\widehat{f}$, hence $S \models \mathbf{FG}\ \neg f$. We address two challenges:

- encode an implicit abstraction of the L2S reduction. The encoding allows us to enumerate, incrementally, abstract lasso-shaped paths $\widehat{\pi}$ satisfying $\widehat{f}$; and

- refine the abstraction when an abstract lasso-shaped path $\widehat{\pi}$ satisfying $\widehat{f}$ cannot be concretized to an infinite, concrete path satisfying f.

While the first challenge is similar to IC3IA, consider the following example for understanding the second issue.

Example 5 ([J7]) *Consider the transition system* $S = \langle \{c, d\}, c = 0 \wedge d \geq 0, (c' = c{+}1 \wedge d' = d) \rangle$, *with* c *and* d *integers, the verification problem* $S \models \mathbf{FG}\ c > d$, *and the set of predicates* $\mathbb{P} := \{c \leq d, 0 \leq d, c = 0\}$. *While* $S \models \mathbf{FG}\ c > d$, *the predicate abstraction of* S *with* $\mathbb{P}$ *admits an abstract lasso-shaped counterexample with a loop on the abstract state* $c \leq d, 0 \leq d, \neg(c = 0)$. *However, such abstract counterexample can be unrolled* i *times (e.g., starting from* $c = 0, d = i + 1$*), similarly to what we shown in Equation* (3.10). *In practice, all the finite prefixes of the above counterexample are all feasible. This means that there are no new predicates that can rule out the abstract counterexample from the L2S abstraction.*

In [C17] we solve the two above problems applying the L2S reduction to find abstract lasso-shaped paths (in the following called $S_{\alpha\text{L2S}}$) and then we extend such encoding to block spurious abstract counterexample with disjunctively well-founded transition invariants [61] (called $S_{\alpha\text{L2S}_\downarrow}$).The $S_{\alpha\text{L2S}}$ encoding $S_{\alpha\text{L2S}} := \langle X_{\alpha\text{L2S}}, I_{\alpha\text{L2S}}, T_{\alpha\text{L2S}} \rangle$ is:

$$X_{\alpha\text{L2S}} := X \cup \{\text{seen}, \text{triggered}, \text{loop}\} \cup \{c_p \mid p \in \mathbb{P}\}, \tag{3.20}$$

$$I_{\alpha\text{L2S}} := I \wedge \neg\text{seen} \wedge \neg\text{triggered} \wedge \neg\text{loop}, \tag{3.21}$$

$$T_{\alpha\text{L2S}} := T \wedge \bigwedge_{p\in\mathbb{P}} c_p \leftrightarrow c'_p \wedge \left(seen' \leftrightarrow \left(seen \vee \bigwedge_{p\in\mathbb{P}} p(X) \leftrightarrow c_p \right) \right) \wedge \tag{3.22}$$

$$(triggered \leftrightarrow (triggered \vee (f \wedge seen'))) \wedge \tag{3.23}$$

$$\left(loop \leftrightarrow \left(triggered' \wedge \bigwedge_{p\in\mathbb{P}} p(X') \leftrightarrow c'_p \right) \right). \tag{3.24}$$

We have that $S_{\alpha\text{L2S}} \models \neg\text{loop}$ iff there exist an abstract lasso-shaped path visiting f infinitely often.

We next introduce well-founded relations. If S is a transition system, let $\mathcal{S}_S$ be its set of reachable states and Q be the state space of S (so, $\mathcal{S}_S \subseteq Q$). A relation $\rho \subseteq Q \times Q$ is a *transition invariant* if it contains the transitive closure of the transition relation T restricted to the reachable states (i.e., $T^+ \cap (R \times R) \subseteq \rho$ [61]). A binary relation $\rho \subseteq Q \times Q$ is well-founded if every non-empty subset $U \subseteq Q$ has a minimal element w.r.t. ρ (i.e., there exists a $m \in U$ such that there are no $u \in U$ where $(m, u) \in \rho$). A relation is *disjunctively well-founded* if it is a finite union of well-founded relations. In the following, let $W(\overline{X}, X)$ be the symbolic representation of a disjunctively well-founded relation expressed over the variables X and $\overline{X}$.

The αL2S$_\downarrow$ encoding $S_{\alpha\text{L2S}_\downarrow} := \langle X_{\alpha\text{L2S}_\downarrow}, I_{\alpha\text{L2S}_\downarrow}, T_{\alpha\text{L2S}_\downarrow} \rangle$ extends the $S_{\alpha\text{L2S}}$ encoding to add

well-founded relations as follows:

$$X_{\alpha\text{L2S}_\downarrow} := X_{\alpha\text{L2S}} \cup \{x_0, \overline{x} \mid x \in X\} \cup \{r, s, w\}, \tag{3.25}$$

$$I_{\alpha\text{L2S}_\downarrow} := I_{\alpha\text{L2S}} \wedge \bigwedge_{x \in X} x_0 = x \wedge r \wedge \neg s \wedge w, \tag{3.26}$$

$$T_{\alpha\text{L2S}_\downarrow} := T_{\alpha\text{L2S}} \wedge \bigwedge_{x \in X} x_0' = x_0 \wedge (w' \leftrightarrow (w \wedge (f \rightarrow r))) \wedge T_{\text{mem}} \wedge T_{\text{check}}, \tag{3.27}$$

$$T_{\text{mem}} := \left((s \leftrightarrow s') \wedge \bigwedge_{x \in X} \overline{x}' = \overline{x}\right) \vee \left(\mathit{seen} \wedge \neg s \wedge s' \wedge f \wedge \bigwedge_{x \in X} \overline{x}' = x\right), \tag{3.28}$$

$$T_{check} := r' \leftrightarrow \left(r \wedge \left((s' \wedge f') \rightarrow \bigvee W(\overline{X}', X')\right)\right). \tag{3.29}$$

We have that if $S_{\alpha\text{L2S}_\downarrow} \models \neg(\text{loop} \wedge \neg w)$ then $S \models \mathbf{FG}\, \neg f$. In the above encoding, we strengthen the safety property of L2S to not consider a violation when the abstract lasso-shaped path *satisfy the transition invariant* W. In practice, the transition invariant W ensures that the lasso-shaped path is not a concrete path.

Refining the abstraction. The algorithm discovers the predicates $\mathbb{P}$ of the abstraction and the transition invariant W using a CEGAR loop. At the beginning of the algorithm, the abstraction predicates are $\mathbb{P} := \emptyset$ and the transition invariants is $W := \bot$. The algorithm checks if $S_{\alpha\text{L2S}_\downarrow} \models \neg(\text{loop} \wedge \neg w)$. If that's the case, then $S \models \mathbf{FG}\, \neg f$, otherwise, the algorithm found an abstract, lasso-shaped path $\widehat{\pi}$ that can be divided in a prefix $\widehat{\pi}_{\text{pref}}$ and a loop $\widehat{\pi}_{\text{loop}}$. The algorithm tries to simulate the abstract path using BMC, encoding a (non-lasso) concrete path of S starting with the abstract prefix $\widehat{\pi}_{\text{pref}}$ followed by a fixed number of unrollings of the abstract loop $\widehat{\pi}_{\text{loop}}$. If the encoding becomes unsatisfiable, then the algorithm use interpolation to find new predicates for the abstraction. If the encoding is satisfiable, then the algorithm strengthen the encoding and tries to find a concrete lasso-shaped path (i.e., a loop). If the encoding is still satisfiable, then there is a concrete lasso-shaped path proving that $S \not\models \mathbf{FG}\, \neg f$. Otherwise, when the algorithm fails finding a concrete lasso-shaped path (e.g., as shown in Example 4), we try to prove that there are no infinite paths that can concretize $\widehat{\pi}$ synthesizing a set of ranking functions. This last concretization uses the following heuristic:

- We enumerate the encoding of a simple lasso program [76] (i.e., a conjunction of predicates from the concretization of $\widehat{\pi}$ and the transition relation T without disjunctions).
- Synthesize a ranking function proving the lasso-path terminates, as shown in [76].
- Use the found ranking function as a disjunctive transition invariant, adding it to W, hence blocking the simple lasso program.

Example 6 ([J7]) *Consider the transition system* $S = \langle \{c, d\}, c = 0 \wedge d \geq 0, c' = c + 1 \wedge d' = d, \top \rangle$*, the LTL property* $\mathbf{FG}\ c > d$*, and the initial set of predicates* $\mathbb{P} := \{c \leq d, c = 0, 0 \leq d\}$*. The algorithm will first find an abstract lasso-shaped counter-example with prefix* $\{x_{c \leq d}, x_{c=0}, x_{0 \leq d}\}$ *and a self loop on the abstract state* $\{x_{c \leq d}, \neg x_{c=0}, x_{0 \leq d}\}$*. The algorithm cannot determine that such abstract counter-example is spurious using bounded model checking, since there is no corresponding lasso-shaped path in* S*. Instead, the algorithm synthesizes a ranking function* $d - c$*, with lower bound* $-1 \leq d - c$*, proving the the abstract loop terminates. The algorithm uses the ranking function to encode a well-founded relation.*

Here, we provided an intuition of the refinement process and we refer to [C17] for more details.

3.3 Related Works

The verification of both invariants and liveness properties for infinite state systems is a wide area of research. Here, we focus on the close verification algorithms for symbolic transition systems expressed with theories, while we ignore a large literature on software verification using SMT (e.g., lazy abstraction [21, 34]).

Lifting SAT-based Model Checking to SMT. There exist several invariant model checking algorithms for transition systems represented symbolically with propositional logic. In particular, SAT-based algorithms (Bounded Model Checking (BMC) [12], k-induction [17], k-liveness [65] , interpolation-based model checking [25], and IC3 [55]) proved to scale well in practice and are the state of the art for hardware verification.All the above algorithms have been extended, in some way, to prove invariants (and other properties) for infinite state systems represented with theories, using SMT instead of SAT. We first discuss k-induction and interpolation-based model checking, and then discuss more in depth the SMT extensions of IC3.

k-induction has been extended to the SMT settings in [26, 59, 85, 88] to alleviate the problem of strengthening the induction hypothesis, which may not be sufficient for proving the inductive step (at any induction depth k) if the system has an infinite number of states. A possible solution to strengthen the inductive hypothesis is to infer an invariant (e.g., via quantifier elimination after a failed proof attempt [26] or with some template-based methods [59]). IC3ia differs from the above k-induction algorithms in several ways. A first difference is that IC3ia tries to prove a property in the abstract (and not concrete) transition system (similarly to IC3ia, k-induction could use implicit abstraction, as shown in [44]). Then, IC3ia tries to find a strengthening for the inductive hypothesis incrementally (i.e., a strengthening to prove relative induction), which is usually an easier task to solve. Furthermore, the performance of k-induction largely depend on the maximum bound k to explore, due to the BMC-like encoding of all the possible paths at length k.

Interpolation based model checking [25, 45, 63, 137] computes an approximation of the reachable states and lifts naturally from the propositional to the first-order logic modulo theory settings. A challenge in interpolation-based model checking is that the size of the interpolants is usually big and the approach is not incremental, differently from IC3 (although some recent works show some promise in using interpolations [143] also for approximating transition relations [144]).

Concrete Extensions of IC3. Several works (e.g., [64, 67, 78, 80, 83, 86, 94]) lifts IC3 to reason on the underlying theories (we will refer to these sets of works as *concrete* instantiations of IC3). In particular, several works follow the framework of Generalized Property Directed Reachability (GPDR) [67], that requires a theory specific generalization (e.g., via an under-approximated quantifier elimination, e.g., see [151]) and a theory specific interpolation. There exists several specialization of GPDR, for example to deal with arithmetic [64, 67, 86], arrays and arithmetic [95], bit-vectors[78, 131], and algebraic data types [147]. An advantage of IC3ia w.r.t. GPDR is to be, in principle, theory agnostic and not requiring an ad-hoc generalization procedure. However, IC3ia is still limited in some cases, for example when the algorithm needs to find an inductive invariant with quantifiers (e.g., as in the case of arrays) or to scale to theories (e.g., NRA) where the satisfiability checking and computing an interpolant may be expensive. Some of these limitations have been addressed with ad-hoc abstraction refinement algorithms (e.g., see [148] for arrays and [105] for non-linear real arithmetic). Moreover, GPDR also solves Constrained Horn Clauses (CHC), a problem that subsumes the invariant verification problem for symbolic transition systems (which corresponds to solving

linear CHC). Here, we do not compare with other existing algorithms that solve CHC (e.g., [58]). An experimental comparison is available from a CHC-COMP competition report [133].

Extending IC3 via Abstraction. The standard predicate abstraction [9] and CEGAR [24] loop can be used to verify infinite-state symbolic transition systems, although incurring in the up-front exponential cost of computing the abstraction. IC3IA uses implicit abstraction [44], which was originally applied to `k-induction`, to avoid such bottleneck. Some approaches [80, 124] follows a similar schema as IC3IA. The CTIGAR algorithm [80] is similar to IC3IA, but it performs both the simulation and refinement eagerly after each generalization of a counterexample to induction and after each relative induction check. Instead, IC3IA performs such steps lazily, only after finding a possible counterexample. The work in [124] extends IC3 with an ad-hoc abstraction for bit-vectors that is determined by the syntax of the underlying system, and not by the bit-vector width.

SMT-based Model Checking Algorithms for LTL verification A prominent approach to prove liveness properties using SAT-based algorithms are liveness to safety reductions [65, 20, 56]. An interesting result is that the liveness to safety reduction of [20] is sound for some classes of infinite state systems [30] (i.e., push-down systems, (ω-)regular model checking, and timed automata). Our extension targets also infinite-state systems where such property does not hold. We borrow the `L2S` reduction for the finite-state predicate abstraction of the system, and then we augment the reduction with well-founded relation to remove abstract loops that do not contain a concrete infinite path violating the property. The `k-liveness` algorithm is sound for infinite-state systems (and, in our work we also use it to to find well-founded relations, see the discussion in [**C17**]), however is limited for infinite-state systems (see our discussion in [**J7**]). It would be interesting to apply similar ideas to `L2S-IA` to the FAIR algorithm [56]. Close to our approach, the idea of refining a predicate abstraction with well-founded relation has been explored in [29]. However, in that approach the refinement adds a monitor and a fairness property, and then uses standard (and explicit) predicate abstraction. A related idea to enumerate easy finite counterexamples first, and then refuting infinite executions only if needed is presented in [92]. Transition predicate abstraction [38] has also been used to prove liveness properties for infinite-state systems.

Chapter 4

Verification of Hybrid Systems with Discrete Abstractions

This chapter presents the work from the following papers: [C13, J6, C25].

In this Chapter we describe how we can reduce both the invariant and liveness verification problems for hybrid systems to the verification of a discrete, infinite-state, transition system. First, in Section 4.1 we focus on the problem of verifying invariants for non-linear dynamical systems using a semi-algebraic abstraction [102], tackling the problem of its exponential up-front computation. Then, in Section 4.2 we provide an algorithm to efficiently compute a relational abstraction for a non-linear system compositionally, using Taylor model flow-pipe construction and a decomposition of the system of differential equations. Finally, Section 4.3 solves the problem of verifying liveness properties (and in practice, LTL properties) for hybrid systems in the presence of Zeno paths. In summary, this Chapter addresses the following research questions:

How can we discretize non-linear dynamics to reduce the verification problem for a hybrid system to the verification problem of a discrete one?

How can we use a discretization to verify liveness properties?

4.1 Implicit Semi-Algebraic Abstraction

In this Section, we focus on the invariant verification problem for polynomial dynamical systems using semi-algebraic abstractions [102]. Consider the verification problem (taken from [C25]) for the dynamical system $\dot{x} = -2y, \dot{y} = x^2$, the set of initial states $I(X) := x - y - \frac{1}{2} \geq 0 \wedge x + 2 > 0$, and invariant property $P(X) := (x+2)^2 + y^2 - 1 > 0$ shown in Figure 4.1a. We can compute a semi-algebraic abstraction for such verification problem fixing a set of polynomials $\mathbb{A} = \{x - y - \frac{1}{2}, x + y + \frac{1}{2}, x + 2\}$. The abstraction partitions the infinite-state space of the dynamical system in a set of discrete regions, one for each possible combination of signs (i.e., $<, >, =$) and polynomials in $\mathbb{A}$. For example, the concretization of the abstract state ① in Figure 4.1b is the state where $x+2 > 0 \wedge x-y-\frac{1}{2} < 0 \wedge x+y+\frac{1}{2} < 0$. Given a set of polynomials $\mathbb{A} := \{a_1, \ldots, a_m\}$, we write the set of abstract states as:

$$3^{\mathbb{A}} := \left\{ \{(a_1, \bowtie_{a_1}), \ldots, (a_m, \bowtie_{a_m})\} \mid \text{for each } a \in \mathbb{A} \text{ and } \bowtie_a \in \{>, <, =\} \right\}.$$

The set $3^{\mathbb{A}}$ is exponential in the number of polynomials $\mathbb{A}$. The abstraction further defines that an abstract state $s_1 \in 3^{\mathbb{A}}$ can transition to another abstract state $s_2 \in \mathbb{A}$ (see the black arrows connecting abstract states in Figure 4.1b) if there exists a concrete state $v_1 \in s_1$ that can reach a state $v_2 \in s_2$ *without visiting another abstract state.* After the finite-state abstraction is built, we can verify if any initial abstract state can reach any bad abstract state with a traversal of the abstract transition system. In Figure 4.1b, we see that all the abstract initial states (i.e., states from ① to ⑥) cannot reach any of the abstract bad states (which we did not drawn in the picture for clarity). In [**C25**], we observe that the semi-algebraic abstraction is, in the end, a predicate abstraction. The main research question we answer is how we can apply the efficient techniques from symbolic model checking, such as computing a predicate abstraction [33] and the IC3IA model checking algorithm via implicit predicate abstraction from Section 3.1 instead of explicitly enumerating the the exponential number of abstract states.

More concretely, we can write a transition relation $T_{\mathbb{A}}(X, X', Z)$ expressing if there is a transition between any pair abstract states (s_1, s_2) in $3^{\mathbb{A}} \times 3^{\mathbb{A}}$ as: [1]

$$T_{\mathbb{A}}(X, X', Z) := \bigvee_{(s_1,s_2)\in 3^{\mathbb{A}}} \Big(s_1(X) \wedge s_2(X') \wedge \neg LZZ_{s_1,\vec{f},s_1\vee s_2}(Z) \Big), \tag{4.1}$$

where, with a slight abuse of notation, given an abstract state s, we write $s(X)$ for its concretization on the variables X (i.e., if $s = \{(a_1, \bowtie_{a_1}), \ldots, (a_m, \bowtie_{a_m})\}$, it's concretization $s(X)$ is $\bigwedge_{(a,\bowtie)\in s} a(X) \bowtie 0$), and $LZZ_{s_1,\vec{f},s_1\vee s_2}(Z)$ is a formula in Non-Linear Real Arithmetic that is satisfiable only if all the trajectories of the system $\vec{f}$ starting in $s_1(Z)$ stays in $s_1(Z)$ when restricted to the $s_1(Z) \vee s_2(Z)$ domain, cannot reach states in $s_2(Z)$ (i.e., $s_1(Z)$ is a *differential invariant* when the state space is restricted to the domain $s_1(Z) \cup s_2(Z)$). The set $Z := \{z_x \mid x \in X\}$ copies and renames the state variables X. The formula $LZZ_{s_1,\vec{f},s_1\vee s_2}(Z)$ has been used in [60] to prove and synthesize differential invariants and, for now, we omit its precise definition. The bottleneck preventing us from using the transition relation $T_{\mathbb{A}}(X, X', Z)$ is that *even expressing* such formula requires to enumerate an exponential number of states. In the following, we show that the transition relation $T_{\mathbb{A}}(X, X', Z)$ can be expressed with a formula that is *linear* in the number of predicates $\mathbb{A}$ and that, furthermore, we can use to model check the abstraction of the dynamical system with IC3IA.

4.1.1 Checking Differential Invariants for Semi-Algebraic Sets

Definition 6 (Differential Invariant) *A formula $\theta(X)$ is a differential invariant [117] for the dynamical system $\dot{\vec{X}} = \vec{f}(\vec{X})$ if for all $\vec{x}_0 \in \mathbb{R}^n$ such that $\vec{x}_0 \models \theta(X)$ and for all $t \geq 0$, $\varphi(\vec{x}_0, t) \models \theta(X)$.*

A sufficient condition to prove that an invariant property holds for a dynamical system is finding a formula $\theta(X)$ such that:

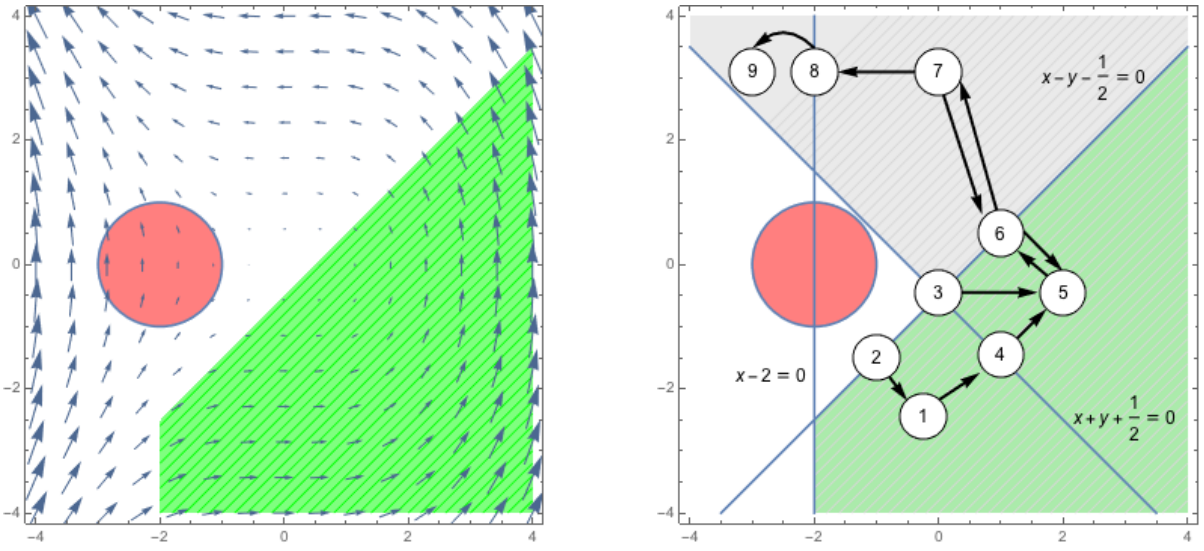
$$\mathrm{Inv}(X) \wedge I(X) \models \theta, \tag{4.2}$$

$$\theta \text{ is a differential invariant}, \tag{4.3}$$

$$\theta(X) \models P(X). \tag{4.4}$$

Intuitively, $\theta(X)$ contains the initial states, is contained in the set if "safe" states, and is invariant under the continuous dynamic of the the system (i.e., once in θ, the system will

[1] For clarity, here we do not include additional constraints in the transition relation, such as the neighborhood relation.



(a) Verification problem and the system's vector field.

(b) Semi-Algebraic abstraction and differential invariant.

Figure 4.1: Example from [C25]. Safety verification problem and reachable states of the abstraction for the non-linear dynamical system $\dot{x} = -2y, \dot{y} = x^2$, invariant $(x+2)^2+y^2-1 > 0$ (the red circle shows the complement of the invariant, $(x+2)^2+y^2-1 \leq 0$), and initial set of states $x-y-\frac{1}{2} \geq 0 \wedge x+2 > 0$ (green region). Figure **(a)** shows the verification problem and the system's vector field. Figure **(b)** shows the reachable abstract states and the transitions of the algebraic abstraction (numbered circles and arrows) computed using an explicit-state algorithm [102] and the differential invariant (green and gray regions) obtained from the set of polynomials $\mathbb{A} = \{x-y-\frac{1}{2}, x+y+\frac{1}{2}, x+2\}$ (blue lines), computed using *Implicit Abstraction*. Abstract states represent different combinations of signs for the abstraction's polynomials. Examples of abstract states are ① $x+2 > 0 \wedge x-y-\frac{1}{2} < 0 \wedge x+y+\frac{1}{2} < 0$, ② $x+2 > 0 \wedge x-y-\frac{1}{2} = 0 \wedge x+y+\frac{1}{2} < 0$, and ③ $x+2 > 0 \wedge x-y-\frac{1}{2} = 0 \wedge x+y+\frac{1}{2} = 0$.

only visit states inside θ). The above proof rule has been widely used to prove invariants for dynamical systems (e.g., see differential dynamic logic [117, 60]).

The LZZ encoding [60] reduces the problem of checking if θ is a differential invariant for the dynamical system $\dot{\vec{X}} = \vec{f}(\vec{X})$ when restricted to the domain η to checking the validity of the formula:

$$\begin{aligned} LZZ_{\theta,\vec{f},\eta}(X) \;:=\; &((\theta(X) \wedge \eta(X) \wedge In_{\vec{f},\eta}(X)) \rightarrow In_{\vec{f},\theta}(X)) \wedge \\ &((\neg\theta(X) \wedge \eta(X) \wedge In_{-\vec{f},\eta}(X)) \rightarrow \neg In_{-\vec{f},\theta}(X)), \end{aligned} \tag{4.5}$$

where $In_{\vec{f},\psi}(X)$ encodes the *inward set* of $\psi(X)$, and $In_{-\vec{f},\psi}(X)$ encodes the *inverse inward set*. Following the explanation from [146], the inward and inverse inward sets are defined as:

$$\mu \models In_{\vec{f},\psi}(X) \text{ iff } \{\vec{x} = \mu[X] \mid \exists \epsilon > 0. \forall t \in (0,\epsilon). \varphi(\vec{x},t) \models \psi(X)\}, \tag{4.6}$$

$$\mu \models In_{-\vec{f},\psi}(X) \text{ iff } \{\vec{x} = \mu[X] \mid \exists \epsilon > 0. \forall t \in (0,\epsilon). \varphi(\vec{x},-t) \models \psi(X)\}. \tag{4.7}$$

Intuitively, $In_{\vec{f},\psi}(X)$ represents the subset of states in $\psi(X)$ that stays in $\psi(X)$ after an infinitesimal amount of time (i.e., "immediately in the future"), while $In_{-\vec{f},\psi}(X)$ is the subset of states in $\psi(X)$ that evolved in $\psi(X)$ for an infinitesimal amount of time in the past. Note that, the interior of a semi-algebraic set $\psi(X)$ is always contained in the (reverse) inward set, so the interesting points of $\psi(X)$ are the one on the boundary.

The Formula (4.5) is valid if for all the states s: (i) if s is in $\theta(X)$, then s must not immediately leave $\theta(X)$; and (ii) if s is in $\neg\theta(X)$, then s must have been in $\neg\theta(X)$.

When all the sets are semi-algebraic we can write the inward sets as a Boolean combination of polynomial equalities and inequalities [60] (i.e., a formula in NRA). The inward set for a predicate $a(X) < 0$, where a is a polynomial is:

$$In_{\vec{f},a<0}(X) \;:=\; a < 0 \vee (a = 0 \wedge a^{(1)}_{\vec{f}}(X) < 0) \vee (a = 0 \wedge a^{(1)}_{\vec{f}}(X) = 0 \wedge a^{(2)}_{\vec{f}} < 0) \vee \ldots \quad (4.8)$$

where $a^{(k)}_{\vec{f}}$ k-th *Lie derivative* of the polynomial $a(X)$ with respect to the ODEs $\vec{f}$ (i.e., $a^{(0)}_{\vec{f}} := a$, and for $i > 0$ $a^{(i)}_{\vec{f}} := \frac{\partial}{\partial \vec{X}} a^{(i-1)}_{\vec{f}} \vec{f}$) Intuitively, $In_{\vec{f},a<0}(X)$ is true for all the states that either are not on the border (i.e., $a < 0$), or are on the border but their vector field "points inward" in the set. However, checking if the vector field points inward in the set is not straightforward. We can check that the vector field points inward on the border (i.e., $a = 0 \wedge a^{(1)}_{\vec{f}}(X) < 0$), but there may be a state such that $a = 0 \wedge a^{(1)}_{\vec{f}}(X) = 0$. Such "vanishing gradient" problem [106] is known, and requires to check all the Lie derivatives until one is not equal to zero to determine if the trajectory will leave $a(X) < 0$ or not. Given a vector field $\vec{f}$ and a polynomial a, there exists an upper bound on the number of consecutive Lie derivatives that can evaluate to 0 for any state in $\mathbb{R}^n$. Given such upper bound, computable via Gröbner bases, the formula (4.8) has a finite number of disjuncts. We can define the inward set $In_{\vec{f},a=0}(X)$ for an equality as $a(X) = 0 \wedge a^{(1)}_{\vec{f}}(X) = 0 \ldots$, and the inward set $In_{\vec{f},\theta}(X)$ for a formula $\theta(X)$ just applying $In_{\vec{f},a \bowtie 0}(X)$ to the predicates $a \bowtie 0$ in the formula θ. We can define $In_{-\vec{f},\theta}(X)$ similarly using $-\vec{f}$ instead of $\vec{f}$. We refer to [60] for a proof that the formula 4.5 is valid only if θ is a differential invariant, and to [146] for proofs of the result that are not restricted to semi-algebraic sets.

4.1.2 Linear Semi-Algebraic Abstraction

In the following, we show how we can write a formula that is equisatisfiable to $T_{\mathbb{A}}(X, X', Z)$ (Equation (4.1)), but that is linear in size in the number of of polynomials in $\mathbb{A}$. We first notice that the LZZ encoding is always applied to check if a single abstract state s_1 is a differential invariant. By applying simple Boolean identities and using the property that the inward set operator distributes over Boolean connectives (see [146]) we have that:

$$\begin{aligned} \neg LZZ_{s_1,\vec{f},s_1 \vee s_2}(Z) &:= ((s_1(Z) \wedge (s_1(Z) \vee s_2(Z)) \wedge In_{\vec{f},s_1 \vee s_2}(X)) \rightarrow In_{\vec{f},s_1}(X)) \wedge \quad (4.9)\\ &\quad ((\neg s_1(Z) \wedge (s_1(Z) \vee s_2(Z)) \wedge In_{-\vec{f},s_1 \vee s_2}(X)) \rightarrow \neg In_{-\vec{f},s_1}(X)) \\ &:= (\neg s_1(Z) \vee \neg In_{\vec{f},s_2}(Z) \vee In_{\vec{f},s_1}(Z)) \wedge \quad (4.10)\\ &\quad (s_1(Z) \vee \neg s_2(Z) \vee \neg In_{-\vec{f},s_1}(Z)). \end{aligned}$$

Now we can distribute the disjuncts of the formula $\neg LZZ_{s_1,\vec{f},s_1\vee s_2}(Z)$ in the formula $T_{\mathbb{A}}(X,X',Z)$ as follows:

$$T_{\mathbb{A}}(X,X',Z) := \exists Z.\ \bigvee_{(s_1,s_2)\in 3^{\mathbb{A}}} (s_1(X)\wedge s_2(X')\wedge \neg LZZ_{s_1,\vec{f},s_1\vee s_2}(Z)) \tag{4.11}$$

$$\Longleftrightarrow \exists Z.\ \bigvee_{(s_1,s_2)\in 3^{\mathbb{A}}} \begin{pmatrix} s_1(X)\wedge s_2(X')\wedge((s_1(Z)\wedge In_{\vec{f},s_2}(Z)\wedge\neg In_{\vec{f},s_1}(Z))\vee \\ (\neg s_1(Z)\wedge s_2(Z)\wedge In_{-\vec{f},s_1}(Z))) \end{pmatrix} \tag{4.12}$$

$$\Longleftrightarrow \exists Z.\ \bigvee_{(s_1,s_2)\in 3^{\mathbb{A}}} \begin{pmatrix} (s_1(X)\wedge s_2(X')\wedge s_1(Z)\wedge In_{\vec{f},s_2}(Z)\wedge\neg In_{\vec{f},s_1}(Z))\vee \\ ((s_1(X)\wedge s_2(X')\wedge\neg s_1(Z)\wedge s_2(Z)\wedge In_{-\vec{f},s_1}(Z)) \end{pmatrix} \tag{4.13}$$

$$\Longleftrightarrow \exists Z.\begin{pmatrix} \bigvee_{(s_1,s_2)\in 3^{\mathbb{A}}} (s_1(X)\wedge s_2(X')\wedge s_1(Z)\wedge In_{\vec{f},s_2}(Z)\wedge\neg In_{\vec{f},s_1}(Z))\vee \\ \bigvee_{(s_1,s_2)\in 3^{\mathbb{A}}} (s_1(X)\wedge s_2(X')\wedge\neg s_1(Z)\wedge s_2(Z)\wedge In_{-\vec{f},s_1}(Z)) \end{pmatrix} \tag{4.14}$$

$$\Longleftrightarrow \exists Z.(InsExpl_{\vec{f}}(X,X',Z)\vee OutExpl_{\vec{f}}(X,X',Z)). \tag{4.15}$$

We are ready to show that there exists a formula $InsSymb_{\vec{f}}(X,X',Z)$ that has a linear size in the number of polynomials $\mathbb{A}$ and that is equivalent to the formula $InsExpl_{\vec{f}}(X,X',Z)$ (and a formula $OutSymb_{\vec{f}}(X,X',Z)$ equivalent to $OutExpl_{\vec{f}}(X,X',Z)$). We expand the definition of the formula $InsExpl_{\vec{f}}(X,X',Z)$ with respect to the predicates in s_1 and s_2. Recall that each concretization of an abstract state is a conjunction of predicates obtained from the set of polynomials $\mathbb{A}$. In the following, if $s=\{(a_1,\bowtie_{a_1}),\ldots,(a_m,\bowtie_{a_m})\}$, we write $a\bowtie 0\in s$ to enumerate the predicates in s. Thus, we write $InsExpl_{\vec{f}}(X,X',Z)$ as: [2]

$$\begin{aligned} InsExpl_{\vec{f}}(X,X',Z) := \bigvee_{s_1,s_2\in 3^{\mathbb{A}}} \Bigg(& \bigwedge_{a\bowtie 0\in s_1} a(X)\bowtie 0\wedge \bigwedge_{a\bowtie 0\in s_2} a(X')\bowtie 0\wedge \\ & \bigwedge_{a\bowtie 0\in s_1} a(Z)\bowtie 0\wedge \bigwedge_{a\bowtie 0\in s_2} In_{\vec{f},a\bowtie 0}(Z)\wedge \\ & \bigvee_{a\bowtie 0\in s_1} \neg In_{\vec{f},a\bowtie 0}(Z)\Bigg). \end{aligned} \tag{4.16}$$

We can express the formula $InsExpl_{\vec{f}}(X,X',Z)$ as a conjunction of the predicates determining the concretization of the abstract states $s_1(X)$ and $s_2(X')$, instead of explicitly enumerating the all the possible abstract states pairs:

$$\begin{aligned} InsSymb_{\vec{f}}(X,X',Z) := & \bigwedge_{a\in\mathbb{A},\bowtie\in\{>,<,=\}} \Big(a(X)\bowtie 0\longrightarrow a(Z)\bowtie 0\Big)\wedge \\ & \bigwedge_{a\in\mathbb{A},\bowtie\in\{>,<,=\}} \Big(a(X')\bowtie 0\longrightarrow In_{\vec{f},a\bowtie 0}(Z)\Big)\wedge \\ & \bigvee_{a\in\mathbb{A},\bowtie\in\{>,<,=\}} \Big(a(X)\bowtie 0\wedge(\neg In_{\vec{f},a\bowtie 0}(Z))\Big). \end{aligned} \tag{4.17}$$

The formula of the linear encoding $InsSymb_{\vec{f}}(X,X',Z)$ is equivalent to the formula $InsExpl_{\vec{f}}(X,X',Z)$. To see this, observe that any satisfying assignment μ (to either $InsSymb_{\vec{f}}(X,X',Z)$ or $InsExpl_{\vec{f}}(X,X',Z)$) assigns a truth value to all the predicates $a(X)\bowtie 0$, $a(X')\bowtie 0$, for all $a\in$

[2] We use De Morgan rules to rewrite the formula $\neg\bigwedge_{a\bowtie 0\in s_1} In_{\vec{f},a\bowtie 0}(Z)$ as $\bigvee_{a\bowtie 0\in s_1}\neg In_{\vec{f},a\bowtie 0}(Z)$.

$\mathbb{A}, \bowtie \in \{>, <, =\}$. Given $\mu \models \mathit{InsExpl}_{\vec{f}}(X, X', Z)$, it's easy to see that $\mu \models \mathit{InsSymb}_{\vec{f}}(X, X', Z)$ just showing that if μ satisfies a unique disjunct in $\mathit{InsExpl}_{\vec{f}}(X, X', Z)$, then μ satisfies $\mathit{InsSymb}_{\vec{f}}(X, X', Z)$. The other direction is similar but uses the simple observation that, whenever $\mu \models a(X) \bowtie 0$, for any $\bowtie' \neq \bowtie$, $\mu \not\models a(X) \bowtie' 0$ (i.e., μ satisfies only "one sign" for each polynomial $a \in \mathbb{A}$). We will skip the proof of the equivalence of the two formulas and the linear version of the formula $\mathit{OutExpl}_{\vec{f}}(X, X', Z)$, which follows a similar construction as the one above, and refer to [0] for the details.

4.1.3 Implicit Semi-Algebraic Abstraction

In practice, we can express the Boolean abstract transition (as in predicate abstraction, see Section 3.1). Let's define the set of predicates $\mathbb{P} := \{a \bowtie 0 \mid a \in \mathbb{A}, \bowtie \in \{<, >, =\}\}$ and the set of Boolean variables $X_{\mathbb{P}} := \{x_{\mathbb{P}} \mid x \in X\}$. Given $\mathit{InsSymb}_{\vec{f}}(X, X', Z)$ and $\mathit{OutSymb}_{\vec{f}}(X, X', Z)$, we can write $T_{\mathbb{A}}(X, X', Z)$ avoiding the explicit enumeration of all the abstract states. The abstract transition relation is:

$$\widehat{T_{\mathit{Symb}_{\mathbb{P}}}}(X_{\mathbb{P}}, X'_{\mathbb{P}}) := \exists X, X', Z. \Big(H_{\mathbb{A}}(X, X_{\mathbb{P}}) \wedge H_{\mathbb{A}}(X', X'_{\mathbb{P}}) \wedge (\mathit{InsSymb}_{\vec{f}}(X, X', Z) \vee \mathit{OutSymb}_{\vec{f}}(X, X', Z)) \Big). \tag{4.18}$$

While $\widehat{T_{\mathit{Symb}_{\mathbb{P}}}}(X_{\mathbb{P}}, X'_{\mathbb{A}})$ represents the Boolean transition relation of the semi-algebraic abstraction, it requires to eliminate the existential quantifiers in X, X', and Z. In practice, we can directly encode the abstract transition relation with implicit predicate abstraction. Let define the following transition relation:

$$T_{\mathit{Impl},\mathbb{P}}(\overline{X}, X', Z) := \mathit{InsSymb}_{\vec{f}}(\overline{X}, X', Z) \vee \mathit{OutSymb}_{\vec{f}}(\overline{X}, X', Z).$$

Then, we can use such transition relation to encode the abstract relative induction check in IC3IA:

$$\begin{aligned} \mathit{AbsRelInd}(\mathcal{F}, T_{\mathit{Impl},\mathbb{P}}, c, \mathbb{P}) := {} & \mathcal{F}(X_{\mathbb{P}}) \wedge c(X_{\mathbb{P}}) \wedge H_{\mathbb{P}}(X, X_{\mathbb{P}}) \wedge H_{\mathbb{P}}(X', X'_{\mathbb{P}}) \wedge \\ & EQ_{\mathbb{P}}(X, \overline{X}) \wedge T_{\mathit{Impl},\mathbb{P}}(\overline{X}, \overline{X}') \wedge EQ_{\mathbb{P}}(\overline{X}', X') \wedge \neg c(X'_{\mathbb{P}}). \end{aligned} \tag{4.19}$$

The above reduction allows us to use implicit predicate abstraction, in general, and also the IC3IA algorithm from the previous Chapter.

4.2 Compositional Relational Abstraction

A *relational abstraction* [62] $R(X, X')$ over-approximates the flow condition of a dynamical system: $D := \langle X, \mathrm{Init}, \mathrm{Inv}, \mathrm{Flow} \rangle$, as follows:

$$\text{for all } t \geq 0 \text{ and states } s, \text{ if } s \text{ can reach } s' \text{ in } t \text{ time in } D \text{ then } s, s' \models R. \tag{4.20}$$

A relational abstraction R is such that if a state s can reach a state s' in the dynamical system, then $s, s' \models R$ (i.e., the formula R expresses a relation between two reachable states s, s').

We can use the relational abstraction R to prove that the dynamical system D satisfies the property $P(X)$ (and more in general to prove liveness properties too, as shown in [39]). We abstract the dynamical system D with a relational abstraction $R(X, X')$ and construct a

discrete transition system $S := \langle X, \text{Init}, R \rangle$ where the transition relation is R. [3] Observing that the relation R contains all the possible pairs of reachable states of D, it's easy to see that:

$$\text{if } S \models P \text{ then } D \models P.$$

Thus, with relational abstraction we reduce the verification problem of a dynamical system to the verification problem of a discrete, infinite-state system.

Remark 4 *The relational abstraction approach to verify invariants extends trivially from dynamical systems to hybrid systems. Given a hybrid automaton H we can obtain a transition system S encoding the automaton's locations, invariants, and discrete transitions. Furthermore, S will have in each location q a "special" time elapse transition labeled with the relational abstraction R_q computed for the flow condition Flow(q) of the location q.*

In the following, we will compute a *time-aware relational abstraction* $R(X, X', t)$ for the dynamical system D:

$$\text{for all } t \geq 0 \text{ and states } s, \text{ if } s \text{ can reach } s' \text{ in } t \text{ time in } D \text{ then } s, s', t \models R(X, X', t). \quad (4.21)$$

The abstraction R is similar to the one from Equation (4.20), but the relation further captures the time t took to reach a state s' from s'.

While there exist techniques to compute relational abstractions for linear dynamics (e.g., [62, 71, **C10**]), they cannot be directly applied to non-linear dynamical systems. In fact, the previous techniques compute a relational abstraction for a linear dynamical system of the form $\vec{f} := A\vec{x} + b$ exploiting either the eigenstructure of the matrix A [62, **C10**], or the explicit solution of the differential equation [71] that is rarely available for non-linear dynamics. A second challenge to compute a useful relational abstraction to verify a property arise from the limitation of the existing VMT algorithms that, at the time of writing of [**J6**], worked only for linear arithmetic theories (the verification algorithm [105] for transition systems expressed in the NRA theory we used in [**C25**] was still not available). For such reason, a technical requirement is to compute a linear (but possibly piecewise) relational abstraction.

In [**J6**], we tackle the computation problem of a relational abstraction for a dynamical system with non-linear dynamics (and furthermore controlled with a time-triggered control systems where a controller runs periodically every Δ time). The main idea of our work is to use Taylor model-based integration [89] over a bounded interval $[0, \Delta]$ to compute a Taylor Model $(a, \mathcal{I})$ approximating the solution φ of the flow condition Flow from 0 to Δ. In practice, such Taylor model is already a relational abstraction: a is a polynomial containing variables from X that computes an approximation of φ bounded by the interval $\mathcal{I}$. However, scaling the computation of such relational abstraction is difficult. First, the the Taylor model integration is not precise when considering a large domain of the state space. Second, the Taylor model integration is expensive when considering a large state space (i.e., a high number of state variables).

We tackle the precision challenge partitioning the state space in intervals and computing a relational abstraction for each one of them. Since computing a precise abstractions via a uniform subdivisions of the state space is expensive, requiring a high number of subdivisions, we propose an *adaptive subdivision* algorithm that partitions the state space non-uniformly, while bounding the precision of the abstraction. We tackle the scalability issue decomposing the system of non-linear ODEs in sub-components (i.e., subsystems of ODEs using a subset of the variables) using the data-dependency relation among state variables, and then computing a relational abstraction for each sub-component in a compositional way (i.e., compute the

[3] We mainly compute R from Flow and over-approximate both Inv and Init conservatively.

abstraction for a component reusing the abstraction of the dependent sub-components). The advantage of the decomposition is that the relational abstraction computation for a sub-component is less expensive, since the sub-component ODEs involve less variables.

In the rest of this Section, we present the computation of time-aware relational abstraction R for a non-linear dynamical system $D := \langle X, \text{Init}, \text{Inv}, \text{Flow}\rangle$ using Taylor models and the adaptive subdivision, and then the compositional computation of the relational abstraction. Here, we simplify our presentation with respect to [J6] to a system D with polynomial dynamic, so the Flow condition expresses a system of ODEs $\dot{\vec{X}} = \vec{f}(\vec{X})$ where $\vec{f}$ is a vector of polynomials. In [J6] we compute a relational abstraction for a non-linear system of ODEs that can include transcendental functions, control inputs, and time-varying disturbances.

4.2.1 Computing Relational Abstractions for Non-Linear Dynamics

A *Taylor Model* (TM) is a pair $(a, \mathcal{I})$ where $a(X)$ is a polynomial (or a vector of polynomials) over some variables X and a domain $\mathcal{M} \subseteq \mathbb{R}^n$, and $\mathcal{I}$ is an interval with the same dimension of X (an interval is a vector $[\vec{a}, \vec{b}]$ where $\vec{a}$ and $\vec{b}$ are vectors in $\mathbb{R}^n$ such that for all $i < n$, $\vec{a}_i \leq \vec{b}_i$). Given a smooth function $f(\vec{x})$ we can compute a TM $(a, \mathcal{I})$ such that, for all $\vec{x} \in \mathcal{M}$, $f(\vec{x}) \in a(x) + \mathcal{I}$. Given a non-linear ODE $\dot{\vec{X}} = \vec{f}(\vec{X})$, the flowmap φ can be approximated with a TM $(a(X, t), \mathcal{I})$ for a bounded time interval $[0, \Delta]$ and an initial condition $\mathcal{M}$ using Taylor model integration [89]. We will compute the TM $(a, \mathcal{I})$ for the time interval $[0, \Delta]$, where Δ is the sampling time of the controller and an initial set of states $\mathcal{M}$. That's it, the time-aware relational abstraction we compute is valid in the interval $[0, \Delta]$ since, after this time, we assume an instantaneous execution of the controller. Thus, the Taylor model $(a, \mathcal{I})$ we compute is such that:

$$\text{for all } x \in \mathcal{M}, \text{for all } t \in [0, \Delta], \varphi(x, t) \in a + \mathcal{I}.$$

In practice, we can write the Taylor model as a formula $R(X, X', t)$ in NRA:

$$\bigwedge_{i<n} \left(a(X, t)_i - \vec{a}_i \leq x_i \leq a(X, t)_i + \vec{b}_i \right).$$

For conciseness, we will write the above formula as $X \in a(X, t) + \mathcal{I}$.

Example 7 ([J6]) *Consider the following system of non-linear ODEs describing a vehicle dynamic:*

$$\begin{aligned} \dot{x} &:= v \sin(\theta), \\ \dot{y} &:= v \cos(\theta), \\ \dot{v} &:= -d_{v(t)} v^2 + u + d_{u(t)}, \\ \dot{\theta} &:= -3\left(\theta - \theta_{ref}\right) + d_{\theta(t)}, \end{aligned} \tag{4.22}$$

where θ_{ref}, u are two control inputs, and $d_{v(t)}$, $d_{u(t)}$, $d_{\theta(t)}$ are bounded disturbances with the following intervals: $d_v \in [0.009, 0.01]$, $d_u \in [-0.45, 0.45]$ (we ignore d_θ in this example). Considering only the velocity v, the TM integration applied to the above ODEs with initial condition $v(0) \in [10, 15]$ and $\Delta = 0.02$ would compute the TM:

$$\begin{aligned} a = {} & v' + 0.45t + tu + 0.176t^2 - 0.0095tv^2 - 0.042t^2v + 0.063t^3 \\ & - 0.0095t^2vu + 0.0033t^2v^2 + 0.019t^3u - 0.0067t^3v + 0.0025t^4. \\ \mathcal{I} = {} & [-0.0025, 0.00238]. \end{aligned}$$

Since the polynomial a can be non-linear in the state variables X, and we want to obtain a linear relational abstraction, we will further compute a *linear truncation* $(a_{L_j}, \mathcal{I}_{L_j})$ of the TM $(a, \mathcal{I})$. The truncation is such that $(a_{L_j}, \mathcal{I}_{L_j})$ is an over-approximation of $(a, \mathcal{I})$ and all the variables X appear in a_{L_j} with degree 1, while t can appear with a higher degree. Such truncation will guarantee that the final relational abstraction $R(X, X', t)$ is expressed in LRA *after substituting* the time variable t with the fixed sampling time Δ.

The above abstraction, while sound, will not be very precise, both because of the size of the initial interval and the linear truncation. Instead, we compute a piece-wise relational abstraction partitioning the state space and then computing a relational for each partition. At the high-level, the algorithm:

- Partitions the space of state and input variables $\mathbb{R}^{|X|}$ in intervals: $\mathcal{M}_1, \ldots, \mathcal{M}_k$.
- For each interval j, computes a TM $(a_j(X, t), \mathcal{I}_j)$ and then its linear truncation $(a_{L_j}(X, t), \mathcal{I}_{L_j})$.
- Obtains a relational abstraction using the intervals and linear truncations as:

$$R(X, X', t) \;:=\; \bigwedge_{j}^{k} \left(X \in \mathcal{M}_j\right) \rightarrow \left(X' \in a_{L_j}(X, t) + \mathcal{I}_{L_j}\right), \tag{4.23}$$

  where, for conciseness, we write $X \in \mathcal{M}_j$ to express that the value of the variables X are included in the interval $\mathcal{M}_j$.

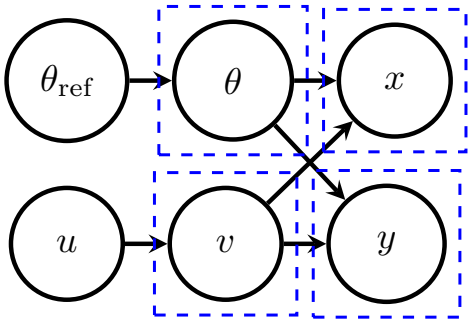
We evaluate the precision of the abstraction $(a_{L_j}(X, t), \mathcal{I}_{L_j})$ with the width $width(\mathcal{I}_{L_j})$ of the interval $\mathcal{I}_{L_j}$ (the width of an interval $[\vec{a}, \vec{b}]$ is $\max(\vec{b} - \vec{a})$). The precision depends on the size and number of intervals $\mathcal{M}_1, \ldots, \mathcal{M}_k$ partitioning the space $\mathbb{R}^{|X|}$. A uniform subdivision of the state space usually requires a large number of small intervals to obtain a precise abstraction (i.e., one with a low width), further requiring to compute several TMs and resulting in a large formula. To cope with such issue, in [**J6**] we propose an algorithm that recursively partitions the state space $\mathbb{R}^{|X|}$, starting with a single interval $\mathcal{M}_j$, computing the TM and the linear truncation $(a_{L_j}(X, t), \mathcal{I}_{L_j})$ for $\mathcal{M}_j$, and evaluating the width $\text{width}(\mathcal{I}_{L_j}) \leq w_{\max}$, for a fixed precision $w_{\max}$. If $\text{width}(\mathcal{I}_{L_j}) \leq w_{\max}$ the algorithm uses $(a_{L_j}(X, t), \mathcal{I}_{L_j})$ as abstraction for $\mathcal{M}_j$. Otherwise, the algorithm splits $\mathcal{M}_j$ uniformly, obtaining a set of new intervals $\mathcal{M}_{j_1}, \ldots, \mathcal{M}_{j_l}$, and then applies the same computation to each one of the new intervals.

Finally, to obtain a time-aware relational abstraction expressed in LRA, we substitute the time t in Equation (4.23) with Δ.

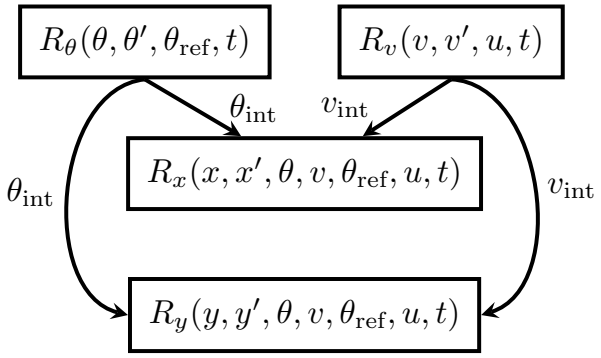
4.2.2 Computing the Abstraction Compositionally

Instead of computing a relational abstraction in one shot for the ODEs, we decompose an ODEs according to the "data dependencies" among the variables [104]. For example, the differential equation of the variable θ from the system of ODEs from Example 7 only depends on the θ variable and the control input θ_{ref}, so we can compute a relational abstraction R_θ that ignores, as an example, the state variable x. A more complex decomposition is the one for the variable x itself, which depends both on v and θ.

Formally, we get all the decompositions for a system of ODEs extracting the strongly connected components (SCCs) of the dependency graph for the ODEs (see Figure 4.2a). The dependency graph is a directed graph where nodes are state or input variables and there is an edge from a variable to another if the source variable is used in the right-hand side of the ODE of the destination variable. When the differential equation for a SCC (e.g., x)



(a) **Dependency graph.** Nodes are input and state variables and there is an edge (a, b) if variable b is used in the right-hand side of $\dot{a}$ in the ODEs. Dotted squares lines represent SCCs for state variables.



(b) **Compositional Relational Abstraction.** We create a relational abstraction for each SCCs in the dependency graph (i.e., R_θ, R_v, R_x, R_v).Since x (and similarly y) depends on θ and v, we compute R_x from x's ODE replacing θ and v with the TM computed to obtain R_θ and R_v (i.e., $\dot{x} := v_{\text{int}} \sin(\theta_{\text{int}})$ with $v_{\text{int}} \in a(v, u, t) + \mathcal{I}_v$ and $\theta_{\text{int}} \in a(\theta, \theta_{\text{ref}}, t) + \mathcal{I}_\theta$.

Figure 4.2: Dependency graph and compositional relational abstraction for the ODEs (4.22).

depends on other components (e.g., θ, v), we use the relational abstraction (in practice, the TM) computed for x to compute the relational abstraction R_θ. First, consider a new differential equation (in practice, a differential inclusion) for the component (e.g., x) where we replace the dependent variables (e.g., θ,v) with an *interface variable* (θ_{int}, v_{int}). Such interface variable is constrained by the TM computed in the abstraction of the corresponding component. For example, if $a(\theta, \theta_{\text{ref}}, t) + \mathcal{I}_\theta$ was the TM computed in the relational abstraction R_θ, we have that $\theta_{\text{int}} \in a(\theta, \theta_{\text{ref}}, t) + \mathcal{I}_\theta$. With such replacement, we compute the relational abstraction R_x compositionally (i.e., using the relational abstraction of the other components) via Taylor Model integration. Observe that the new composition only involves the interface variables (and the constant time inputs).

In practice, a relational abstraction for a variable is a piece-wise function (see Equation (4.23)). This results in the computation of a new TM for the cartesian product of all the intervals (i.e., the M_j-s in Equation (4.23)). Such cartesian product leads to an explosion in the number of relations to compute. In [J6], we propose two optimizations to mitigate such explosion. First, we compute a specific relational abstraction to use in the composition that is precise enough on the interface variables, while ignoring the remaining ones. Such optimization potentially allow us to use a coarser partitioning and hence limits the explosion in the number of intervals. Second, due to the decomposition some non-linear ODEs become linear. For example, the differential equation $\dot{x} := v_{\text{int}} \sin(\theta_{\text{int}})$ is linear in x, so we can use the explicit solution $x(t) := x(0) + \int_0^t v(s)cos(\theta(s))ds$ to compute a TM for x. In this case, we can use the optimized computation of the time-aware relational abstraction for linear systems.

4.3 K-Zeno: Extending k-liveness for Verifying Hybrid Systems

The k-liveness algorithm [65] reduces the liveness verification problem of $S \models \mathbf{FG}\ \neg f$ to a sequence of invariant verification problems. The main observation of the k-liveness algorithm is that, if the number of times S visits f in *all the possible executions* is bounded by a natural number K, then $S \models \mathbf{FG}\ \neg f$. In fact, when such bound exists any infinite path σ of S has a finite prefix where f holds in some state at most K times, but after such prefix f cannot

be visited anymore (i.e., $\sigma^i \models \mathbf{G}\ \neg f$ for some $i \in \mathbb{N}$). Formally, we have that:

$$\text{if } \exists K \in \mathbb{N} \text{ such that } S \models \#(f) \leq K, \text{ then } S \models \mathbf{FG}\ \neg f, \tag{4.24}$$

where $\#(f)$ counts the maximum number of times f was true in all the paths. The k-liveness algorithm reduces the verification problem to a sequence of invariant verification problems checking the above condition for a fixed value of K (e.g., $S \models \#(f) \leq 0$, $S \models \#(f) \leq 1, \ldots$), stopping when there exists $K \in \mathbb{N}$ such that $S \models \#(f) \leq K$. While k-liveness can use any invariant verification algorithm to check an invariant problem, a practical advantage of the approach when using the IC3 model checking algorithm is that the sequence of invariant verification problems can be solved incrementally. In practice, one can reuse the frames of the IC3 algorithm computed checking $S \models \#(f) \leq i$ when verifying $S \models \#(f) \leq i+1$ (observe that S is the same in all the invariant problems). When the system has a finite number of states, we have that the other direction of the implication in Equation (4.24) holds (i.e., if $S \models \mathbf{FG}\ \neg f$, then there exists $K \in \mathbb{N}$ such that $S \models \#(f) \leq K$), so the algorithm will terminate if $S \models \mathbf{FG}\ \neg f$. Observe that k-liveness will not terminate if $S \not\models \mathbf{FG}\ \neg f$ (in practice, we can run BMC and k-liveness in lock-step to ensure termination).

Here, we want to apply the k-liveness algorithm to prove liveness properties on hybrid systems. In the following, we assume the hybrid automaton H has a the transition system encoding S_H guaranteeing that:

$$H \models \mathbf{FG}\ \neg f \text{ iff } S_H \models \mathbf{FG}\ \neg f, \tag{4.25}$$

where f is a quantifier free formula in the real valued variables X of H and in a special variable loc identifying the current location (e.g., loc $= l$ evaluates to true in a state (l_0, s_0) if the automaton is in location l_0) . We will also assume that the same variable loc exists in the encoding S_H. Note that such encoding S_H is an over-approximation of H, as the ones we computed in the previous sections. We further assume the existence of a variable time that keep track of the total time elapsed in the automaton execution (i.e., time is a clock that is never reset). Thus, we can apply k-liveness to prove $S_H \models \mathbf{FG}\ \neg f$. Checking liveness properties for continuous systems requires to exclude *Zeno paths*: an infinite path of a hybrid automaton is non-Zeno if the time elapsed in the path diverges. While several techniques assume all infinite paths are non-Zeno, such condition is difficult to impose (especially when starting from a system including different components). Instead, we can directly ignore Zeno paths in the model checking problem. [4]

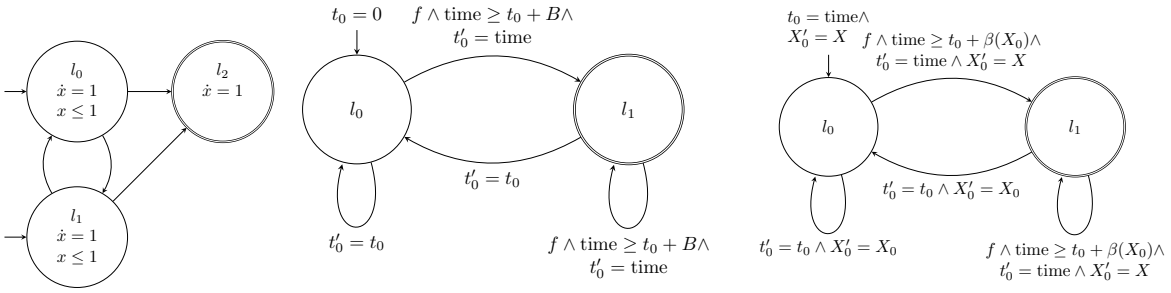
k-liveness cannot be applied successfully when the underlying hybrid automaton has Zeno paths. To see this issue, consider the timed automaton of Figure 4.3a. While the automaton satisfies the property $\mathbf{FG}\ \neg\text{loc} \neq l_2$, it has a non-Zeno path alternating the locations l_0 and l_1 where time cannot diverge (i.e., due to the invariant $x \leq 1$, time cannot be greater than 1). Applying k-liveness on the encoding of this timed automaton (i.e., $S_H \models \mathbf{FG}\ \neg\text{loc} \neq l_2$) would fail, since for each choice of $K \in \mathbb{N}$ we have that $S \not\models \#(f) \leq K$.

In [**C13**] we provide a solution for the problem with the K-Zeno algorithm. K-Zeno introduces a monitor automaton $Z_B(f, \text{time})$, shown in Figure 4.3b, that is composed with S_H and changes the fairness condition to reach the accepting state of the monitor l_1 so that:

$$S_H \models \mathbf{FG}\ \neg f \textit{ iff } S_H \times Z_B \models \mathbf{FG}\ \neg(Z_B.\text{loc} = l_1), \tag{4.26}$$

where $(Z_B.\text{loc} = l_1)$ holds in all the states of the products where the location of the monitor Z_B is l_1. The monitor Z_B only accepts paths where the occurrences of the fairness condition f are separated by at least B time. In Z_B, the discrete variable t_0 saves the first time f was true.

[4] Technically, we should consider a state reachable only if it is on a non-Zeno paths.



(a) Timed automaton satisfying **FG** ¬loc ≠ l_2

(b) Monitor $Z_B(f, \text{time})$

(c) Monitor $Z_B(f, \text{time}, X)$

Figure 4.3: K-Zeno monitors and simple timed automata

Then, two conditions must hold to reach l_1: (i) f must hold; and (ii) $time > t_0 + B$. `k-liveness` would succeed in proving $S_H \times Z_B \models \mathbf{FG}\ \neg Z_B.\text{loc} = l_1$ with $B = 1$ and $K = 1$. Note that a similar monitor to Z_B is used in Uppaal to remove Zeno paths. The monitor Z_B, however, does not help for hybrid systems and for parametric timed and hybrid systems (i.e., a parameter is a variable that is assigned in the initial state and never changed afterwards). K-Zeno introduces the more generic monitor automaton $Z_\beta(f, \text{time}, X)$ shown in Figure 4.3c where the bound $\beta(X_0)$ is not a constant, but an expression that depends on the "history" variables X_0. In such case, the monitor accepts a path if it visits the instances of the fairness signal f that are distant more than $\beta(X_0)$ time infinitely often. In [**C13**], we show that there exists an expression for $\beta(X_0)$ that guarantees the relative completeness of K-Zeno for Parametric Rectangular Hybrid Automata (PRHA) that are initialized and have bounded non-determinism. For such subclass of systems, the K-Zeno algorithm is both sound and *relative complete* in proving liveness properties (i.e., the algorithm is complete assuming every invariant verification check terminates).

4.4 Related Works

The verification algorithms for dynamical and hybrid systems can be categorized in set-based reachability analysis, abstraction-based verification, and logic-based verification [51, 113, 123]. All these families of techniques have strengths and weaknesses and are complementary, being able to solve specific problems for particular classes of hybrid systems. We will discuss the closest related works for the contributions of this Chapter.

Abstraction-based verification. Several works propose to abstract the state space of a hybrid system. For example, the invariant verification of piece-wise hybrid systems [40, 145] can be reduced to the VMT problem in LRA. In those simple settings, the IC3IA algorithm is competitive in proving invariant properties for piece-wise hybrid systems [**C22**]. Other abstraction techniques have been applied to linear [40] and non-linear [10] dynamics where, in particular, hybridization [48] is a popular technique to partition the state space of a non-linear hybrid system.

The implicit semi-algebraic abstraction of Section 4.1 is a qualitative abstraction [43, 46] and focuses on the unbounded time invariant verification problem for polynomial dynamical systems. The approach is also relevant as sub-procedure for proving invariant for hybrid programs [42] with Keymaera X [93] or for hybrid CPS with the HHL Prover [97] (and the evalu-

ation in [C25] considers verification problems from the automated theorem proving domain). The previous computations of qualitative abstractions (e.g., [43, 66, 91] did not compute the *exact* abstraction for a dynamical system, but an approximation, possibly loosing precision. The semi-algebraic abstraction has been computed explicitly in [102, 101] and implemented in the Pegasus [129] tool. In [C25], we show that the explicit abstraction computation often does not scale, while implicit semi-algebraic abstraction does. In [102], the authors also proposes an optimization called *DWCL* that recursively finds inductive invariants among the abstraction predicates before computing the abstraction. Intuitively, such approach reduces the abstract state space (i.e., ruling out the complement of the invariant predicate). *DWCL* is somehow an orthogonal approach to the implicit semi-algebraic abstraction.

Relational abstractions [39, 62, 71, 69, C10, 96] abstracts the dynamical system's trajectories with a discrete transition relation, reducing the verification problem if the continuous system to a verification problem on a discrete system. While relational abstraction is a general concept, practical approaches mainly focused on abstracting linear systems, either via template-based methods [62], reachability analysis [39, 71], or exploiting the eigenvalues and eigenvector of the dynamical system [69, C10]. In Section 4.2 we propose an approach that use (time bounded) reachability analysis to compute relational abstractions, as [71], but via Taylor model computation and for non-linear dynamical systems (including transcendental functions and time-varying disturbances). In some way, we can see semi-algebraic abstraction as a relational abstraction: in fact, the transition relation $T_{\mathbb{A}}(X, X', Z)$ (and also it's implicit abstraction version) can be seen as a relational abstraction over-approximating all the continuous trajectories from X to X'. Since relational abstractions can be easily composed (e.g., just intersecting the relations [C10]), the two abstraction techniques could be composed together to obtain a more precise abstraction.

A common problem in abstract verification is finding a precise enough abstraction to prove the property of interest. Counter Example Guided Abstraction Refinement (CEGAR) has also been applied to hybrid systems [23, 31, 108, 103, 120] for automatically refining an abstraction. Simulation and refinement are an open problem both for relational abstraction and the semi-algebraic abstraction. While we can improve the precision of relational abstraction [J6], finding new polynomials to refine the semi-algebraic abstraction is more challenging.

Set-based reachability analysis. Set-based reachability analysis is one of the prominent approaches for the bounded time verification of dynamical and hybrid systems, dealing with linear and non-linear dynamics (see [135] for a recent survey). Usually, set-based reachability analysis are not adequate for proving invariant properties, since finding a fixed-point is difficult and requires some widening operator. We use set-based reachability analysis [74] to compute relational abstractions. Similarly to our settings, other works use reachability analysis as a primitive in algorithms that prove properties. For example, [109] uses reachability analysis for building a barrier certificate, and [128] uses reachability analysis in a proof rule to prove a stability property.

HySAT [41] and dReal [75] implements ad-hoc decision procedures for differential equations in Satisfiability Modulo Theory (SMT) solvers using interval arithmetic (and reachability analysis for dynamical systems) as underlying theory solvers. The approach is very expressive and successful in finding violations (e.g., via BMC). In the same spirit of the first research problem presented in this thesis, an open question is to extend the efficient SAT-based algorithms to use such SMT solvers effectively.

Logic-based verification Several approaches uses constraints to find certificates proving a property of interest for hybrid systems. A prominent approach is this category is to find

certificates of safety using template-based methods. A well known approach are barrier certificates [35, 77, 141]. A difference between semi-algebraic abstraction and barrier certificate is that, usually, barrier certificates are a single polynomial and are found via numerical optimization solvers (which require a further validation check [**C26**] that may fail). Finding a single polynomial as certificate may be difficult, especially when dealing with hybrid systems. A basic result for finding certificates for dynamical systems that characterize positive invariant sets [13] is the Nagumo theorem [1], which works for closed sets. The LZZ algorithm [60] that we use when encoding the semi-algebraic abstraction is a general invariant synthesis procedure that can use an arbitrary semi-algebraic set as template (i.e., a formula in the NRA theory). However, finding an invariant from a template usually does not scale, requiring to eliminate several quantifiers from the formula. Instead, we use LZZ as a decision procedure (i.e., we do not have parameters to find in our case), removing the need of quantifier elimination. Other approaches find invariants of the dynamical system, for example first integrals [19] or Darboux Polynomials [82]. Such invariants can be used directly as polynomials of the abstraction to prune the state space as done in the *DWCL* algorithm.

Zeno Paths in Temporal Logic Verification. There exist several approaches for verifying temporal logic specifications for timed and hybrid systems (e.g., [114, 90, 73, 52, 118, 134]). In the context of hybrid systems, several techniques focus on falsification (e.g., [52, 49]), while the K-Zeno algorithm focuses on unbounded time verification. Moreover, K-Zeno verifies LTL properties evaluated at a discrete time in the hybrid trace, differently from logic capturing intervals (e.g., Signal Temporal Logic [28], Metric Interval Temporal Logic [37]) or the continuous evaluation of predicates such as HRELTL [90]. In some settings, K-Zeno can be applied to such logics after and additional reduction (e.g., see [122]). Removing Zeno paths from the verification results is a known problem for timed automata that can be solved with a simple monitor automaton forcing the divergence of time (e.g., see [14]). However, such monitor is not sufficient for hybrid systems and timed systems with parameters. K-Zeno constructs a monitor that is sufficient to remove Zeno path for some subclasses of hybrid automata.

Chapter 5

Other Research Activities

In this Section, we briefly summarize the other main research activities carried out after the award of the Ph.D. degree in 2014.

5.1 Formal Analysis of Switched Kirchhoff Networks [C16, C18, C19]

Cyber-Physical Systems (CPSs) are often designed using acausal modeling, which is prominent when designing electric, hydraulic, mechanical systems, and their combination, and is the paradigm used in design languages such as Modelica. In acausal modeling, a network of components are connected together through terminals in a node. In practice, each terminal represents two physical quantities called *effort* and *flow* (e.g., for the electric domain the effort is the potential while the flow is the current). Then, the physical quantities on the nodes behave according to the *Kirchhoff Laws*: all the efforts of the terminals in the same node must be equal and the sum of all the flows of the node is zero. Thus, differently from the casual modeling used in languages like Simulink, in acausal modeling terminals *do not specify* an input/output direction. Each component specifies the continuous dynamic locally referring to the flow and effort of its terminals and the global dynamic of the system is a system of Differential Algebraic Equations (DAEs) obtained adding to the differential equations of the components the equations of the Kirchoff Laws for the network's node. In our work, we focus on acausal models that introduce *discrete switches* in the components (i.e., they are a hybrid system). We call such models *Switched Kirchhoff Networks* (SKMs).

There are two important challenges preventing the verification of properties for a SKM: (i) the dynamic of the system is described with a system of *Differential Algebraic Equations* (DAEs), which is difficult to verify; and (ii) a SKM has a different system of DAEs *for each possible configuration* of the switches of the network (i.e., each configuration corresponds to a location of a hybrid system). In [C16, C18], we solve the following problems: (1) Checking if every configuration of a SKM can be written as a system of ODEs (*validation problem*). If the check succeeds, then each DAEs is an implicit representation of a system of ODEs; and (2) Translating the SKM as a hybrid automaton (*reformulation problem*) that we can verify (e.g., with the abstraction techniques presented in Chapter 4).

The main challenge we solve is that the number of configurations to check and reformulate is exponential in the number of switches, so an explicit enumeration usually does not scale.We tackle such issue reducing the validation problem to a sequence of satisfiability checks of LRA formulas. Similarly, we reformulate the network in a hybrid automaton symbolically. The reformulation algorithm avoids the exponential blowup grouping together configurations that have the same system of differential equations. The algorithm is similar to an AllSAT enumeration of all the network configurations: the algorithm finds the ODEs for a configuration, synthesizes all the configurations with the same ODEs (using quantifier

elimination), and then "blocks" all of them from the following iterations of the enumeration. In [C16, C18], we show that the symbolic algorithms scale to validate and reformulate SKMs from different domains (e.g., RLC circuits, a landing gear system, a wheel breaking system) and in [C19] we show that the methodology can verify properties on an industrial railway signaling system implemented via electrical relays.

5.2 Event-Driven Program Verification and Synthesis [C20, C21, C23, J8]

Event-Driven Programs are programs (called *app* in the context of mobile software) that implement callback functions to react to events from a *framework* (i.e., the framework invokes the *callback* function in the app, which does not have its own entry point). What is difficult about programming event-driven programs is that the framework's code that defines the program's control flow, including the order of execution of the app's callbacks, is often unavailable and too complex to understand. [1] In practice, app developers rely on the ambiguous and incomplete, at least regarding the order of callbacks' execution, Android documentation. Due to this situation, developers often introduce bugs that only appears under a specific sequence of callback execution and for such reason are difficult to understand. This same challenge is present when analyzing statically the app code. The classic approaches to static analysis for event-driven programs consist of eagerly modeling the missing framework code (e.g., see [79, 87]), and then composing it with the app code to have a complete program. What is insufficient of the eager modeling when proving assertion is that: (i) a model only covers the most popular subsets of the framework's classes; and (ii) a model is, unfortunately, often unsound [C23].

In our work, we ask how can we analyze such event-driven programs without providing a detailed model of the framework.

In [C23] we first provide a clean formalization of an event-driven programs that identifies the framework/app interface as the sequence of call invocations (and call return) between the app and the framework (i.e., a sequence of *messages* that we call *message histories*). Given such interface, we show that we can specify the *callback control flow* (i.e., the order of execution of the callbacks) imposed by the framework with a specification language over messages. We further show how we can validate automatically such specifications against real execution traces (i.e., similarly to conformance testing), increasing the confidence of their soundness. In [C21], we show how one can efficiently synthesize, using an active learning algorithm, a subset of such callback control flow specifications. What is difficult in applying active learning to real event-driven programs is the implementation of the membership queries, since the framework is a black-box. We show how to implement such queries efficiently via a *bounded sequence* of membership queries. More recently, in [J8] we tackle the invariant verification problem for event-driven programs. We observe that, in order to prove an invariant property in an app (e.g., proving that the program cannot reach a *Null Pointer Exception* at a specific program location), we do not need to eagerly specify a framework model. Instead, we show that one can specify a limited set of message histories, expressed with a restricted temporal logic, and still being able to prove invariant properties of interest. We develop a *goal-directed* [81] abstract interpretation algorithm that reason on both the app executions

[1] Here, we assume that statically analyzing the composition of the app and framework code is not feasible, as it is the case of Android (while being open source, the Android code is huge and mixes Java and native code). Furthermore, in this work we focus on programs written in a high-level language (e.g., Java) and with dynamic memory allocation, as opposed to the programs we considered in Chapter 3 that were from the embedded software domain (e.g., programs with complex logic and numerical computations, but usually no recursion, memory allocation, and library calls).

and message histories and can prove assertions in an app when only providing a limited set of framework specifications.

5.3 Abstractions in Hierarchical Reinforcement Learning [C27, C28]

Reinforcement Learning (RL) [11] learns to control an agent interacting with an *unknown* environment to achieve a specific task. A RL algorithm learns a policy $\pi(s, a)$, mapping a state s and an action a to the probability of taking a in the state s. In practice, the agent samples the "best" action to execute in a state s_t at time t using π. The action will have an effect on the environment and the agent will observe the next state (i.e., s_{t+1}) and a reward r_{t+1}, a number quantifying the success of the agent in accomplishing the task. RL tries to learn an optimal policy π that maximizes the expected cumulative reward. [2]

Applying RL to environments that are highly-dimensional and continuous, like the ones seen in robotics [70], is difficult. In the above settings, a promising family of RL approaches are *Hierarchical Reinforcement Learning* (HRL) [149], where a high-level agent learns to select sub-goals in a *goal space* (e.g., the goal space could be directly the state space of the agent) and then a low-level agent learns how to achieve each sub-goal selecting actions of the agent. In such hierarchy, the high-level agent learns a more abstract policy (e.g., in a maze, the high-level agent may learn to select positions to reach the maze's exit) and works at a higher "resolution" (e.g., select a goal every k times), while the low-level agent learns to solve parts of the more abstract tasks (e.g., move the agent from a point to another in the maze, ignoring the path to the exit).

The scalability of the above HRL algorithm greatly depends on the choice of the goal space the high-level agent uses (i.e., the mapping from the observation space to a goal space). However, efficient *goal space representations* are not known a-priori [125] and depend on the environment and solved task. What is challenging about learning goal space representations is to learn a representation that "preserves the underlying environment's dynamics" [125].

We tackle the goal space representation learning problem in [C27] using the concepts of abstraction. We propose to learn an abstraction function mapping a set of observed states to an *abstract* goal state, similarly in spirit to finite-state abstractions used in verification. The abstraction maps in the same goal state all the observed states that "behave similarly". We formalize such notion with the reachability relation among states, similarly to [153], but using abstractions: two observed states are abstracted in the same abstract state if they both reach the same set of abstract states. [3]

We provide a HRL algorithm that learns, at the same time, a hierarchical policy and a goal space representation. The main innovation of the algorithm is to refine an initial abstraction after every major learning iteration (i.e., an episode), resembling in a way a CEGAR loop (however, here we refine the abstraction when the learning algorithm does not make any significant progress and still cannot solve a task). The main technical challenge in the refinement is that the (unknown) reachability relation is approximated from the observed data via a neural network. We use reachability analysis [139] to analyze each abstract goal state, then we split an abstract state in two new states if they "behave differently" (i.e., reach different abstract states).

In [C28], we extend the algorithm from [C27] to scale on high dimensional environments. The extension adds an intermediate layer in the hierarchy, implementing a mechanism called temporal abstraction [116]. In the experimental evaluation we show that learning a goal representation as an abstraction allow the HRL algorithm to efficiently learn a policy and

[2] Here, we provide a high-level intuition of RL, avoiding its formalization as Markov Decision Process.
[3] The idea is, in spirit, similar to a partition refinement algorithm.

outperforming the existing state of the art [116, 153, 138] when the number of dimensions to consider increases. In [**C28**], we further provide a theoretical analysis providing some optimality guarantees on the learned policy.

Chapter 6

Conclusions and Future Directions

This manuscript synthesizes my contributions to the VMT and the hybrid system verification problems.

The first contribution, IC3IA [**C12**], tackles the problem of efficient invariant verification for infinite-state systems represented with theories. Despite it's simplicity, IC3IA demonstrated to work extremely well in practice: for example, the experimental results in [**J5**] show that IC3IA is efficient and, interestingly, pushes the application of predicate abstraction verifying abstractions with hundreds of predicates, which are out of reach when computing the abstraction explicitly. Some experiments in the competition for Horn Clause verification [132] shows that IC3IA is competitive with more recent algorithms (for verifying linear horn clauses in the LRA theory). Another interesting application of IC3IA was in the verification of piece-wise constant hybrid automata, being competitive in proving invariants with ad-hoc verification techniques [119]. More importantly, IC3IA was the enabler for solving other verification problems for infinite-state systems. In fact, there has been multiple extensions using IC3IA to verify liveness properties (both the L2S-IA and K-Zeno algorithm we presented are tightly integrated with IC3IA and run IC3IA incrementally), verify more "challenging" theories [105, 148], and, as we shown in this thesis, verifying invariant for hybrid systems. Such trend is not surprising, since invariant verification algorithms are often a key component when solving more complex problem (e.g., synthesis [72, **C9**, 98], safety assessment [99], and verifying hyper-properties [47]). Moreover, the IC3IA algorithm is the main infinite-state verification algorithm of the model checker *nuXmv*, has been one of the main verification backend of different tools (e.g., XSAP [99], HyCOMP [**C15**], Kratos2 [152]), and has been re-implemented in other tools (e.g., [115, 140]).

Apart from extending IC3IA to new theories, an interesting research direction is model checking hyper-properties [47], which generalize trace properties to sets of traces. Hyper-properties allow us to express important properties of the system, especially related to security (e.g., non-interference, non-determinism, ...). While there has been a lot of progress in model checking hype-properties recently (e.g., see [142, 127, 150]), verifying hyper-properties for infinite-state systems is still a less explored problem. In particular, the verification of k-safety properties for infinite-state systems has been more explored (e.g., self-composition [53]), while the verification of liveness hyper-properties for infinite-state systems seems a more open problem. The IC3IA and L2S-IA algorithms provide a solid start to tackle the problem, which is challenging due to the quantifications on traces existing in the property.

The hybrid system verification techniques we summarized in this thesis also use the framework of abstraction for solving the challenge of analyzing the system's continuous dynamics (while in IC3IA, the main use of abstraction was to tackle the infinite-state space). The semi-algebraic abstraction can be used to verify dynamical (and, in principle, hybrid) systems with polynomial dynamics that are used in the Keymaera theorem prover [93]. The

main contribution of [C25] is to avoid the explicit abstraction computation, which is expensive, using symbolic model checking techniques. The experimental evaluation in [C25] shows that the algorithm is competitive with the existing techniques (e.g., [102]) and, in practice, complementary. However, the algorithm is still limited by the scalability of the underlying model checking algorithms for the NRA theory and the need of finding an adequate set of polynomials for the abstraction.

The compositional relational abstraction technique also showed [J6] some promising result in verifying relatively complex time-triggered controllers for a class of non-linear systems (including transcendental functions). However, a limitation of the approach is the tradeoff between the precision and the size of the abstraction. In particular, the relational abstractions we compute can be difficult to model check: one issue is that the underlying rational constants in the model, coming from the Taylor model approximations, are expensive to represent using infinite-precision arithmetic; another issue is that the model checker needs to explore a high number of steps in the abstraction, and symbolic techniques suffer in such situation (e.g., an interesting avenue is to explore acceleration techniques, e.g., [84]).

An obvious limitation of both our abstraction techniques is the lack of an automatic refinement of the abstraction. Providing the right polynomials for the semi-algebraic abstraction is quite difficult, as providing the right precision for the compositional relational abstraction. This situation contrasts with the progress in VMT, where the CEGAR approach allows us to have a push button algorithm. The challenges already starts with the need of simulating the abstract counterexample (observe that this task is quite challenging in the context of the semi-algebraic abstraction). Then, the other challenge is finding the "right" polynomials for the refinement, which requires to develop some novel techniques different from interpolation.

While the K-Zeno algorithm demonstrated to be effective on a number of benchmarks, it's applicability to hybrid systems with more complex dynamics is still limited and would require to discover different symbolic bounds, for example using Lyapunov function.

Finally, also in the case of hybrid system verification, the extension to model check hyper-properties is a possible research direction, especially focusing on systems with "simple" dynamics (even timed automata) first, which would allow us to reuse the techniques developed for the discrete, infinite-state case (some works already exists for hyper-properties and hybrid system, see for example [107, 111, 126]).

Publication List

Conference Publications

[C1] Roberto Cavada, Alessandro Cimatti, Alessandro Mariotti, Cristian Mattarei, Andrea Micheli, Sergio Mover, Marco Pensallorto, Marco Roveri, Angelo Susi, and Stefano Tonetta. "Supporting Requirements Validation: The EuRailCheck Tool". In: *ASE*. IEEE Computer Society, 2009, pp. 665–667.

[C2] Lei Bu, Alessandro Cimatti, Xuandong Li, Sergio Mover, and Stefano Tonetta. "Model Checking of Hybrid Systems Using Shallow Synchronization". In: *FMOODS/FORTE*. Vol. 6117. Lecture Notes in Computer Science. Springer, 2010, pp. 155–169.

[C3] Alessandro Cimatti, Sergio Mover, Marco Roveri, and Stefano Tonetta. "From Sequential Extended Regular Expressions to NFA with Symbolic Labels". In: *CIAA*. Vol. 6482. Lecture Notes in Computer Science. Springer, 2010, pp. 87–94.

[C4] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. "Efficient Scenario Verification for Hybrid Automata". In: *CAV*. Vol. 6806. Lecture Notes in Computer Science. Springer, 2011, pp. 317–332.

[C5] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. "HyDI: A Language for Symbolic Hybrid Systems with Discrete Interaction". In: *EUROMICRO-SEAA*. IEEE Computer Society, 2011, pp. 275–278.

[C6] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. "Proving and explaining the unfeasibility of message sequence charts for hybrid systems". In: *FMCAD*. FMCAD Inc., 2011, pp. 54–62.

[C7] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. "A quantifier-free SMT encoding of non-linear hybrid automata". In: *FMCAD*. IEEE, 2012, pp. 187–195.

[C8] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. "SMT-Based Verification of Hybrid Systems". In: *AAAI*. AAAI Press, 2012, pp. 2100–2105.

[C9] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. "Parameter synthesis with IC3". In: *FMCAD*. IEEE, 2013, pp. 165–168.

[C10] Sergio Mover, Alessandro Cimatti, Ashish Tiwari, and Stefano Tonetta. "Time-aware relational abstractions for hybrid systems". In: *EMSOFT*. IEEE, 2013, 14:1–14:10.

[C11] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. "The nuXmv Symbolic Model Checker". In: *CAV*. Vol. 8559. Lecture Notes in Computer Science. Springer, 2014, pp. 334–342.

[C12] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. "IC3 Modulo Theories via Implicit Predicate Abstraction". In: *TACAS*. Vol. 8413. Lecture Notes in Computer Science. Springer, 2014, pp. 46–61.

[C13] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. "Verifying LTL Properties of Hybrid Systems with K-Liveness". In: *CAV*. Vol. 8559. Lecture Notes in Computer Science. Springer, 2014, pp. 424–440.

[C14] Simon Bliudze, Alessandro Cimatti, Mohamad Jaber, Sergio Mover, Marco Roveri, Wajeb Saab, and Qiang Wang. "Formal Verification of Infinite-State BIP Models". In: *ATVA*. Vol. 9364. Lecture Notes in Computer Science. Springer, 2015, pp. 326–343.

[C15] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. "HyComp: An SMT-Based Model Checker for Hybrid Systems". In: *TACAS*. Vol. 9035. Lecture Notes in Computer Science. Springer, 2015, pp. 52–67.

[C16] Alessandro Cimatti, Sergio Mover, and Mirko Sessa. "From Electrical Switched Networks to Hybrid Automata". In: *FM*. Vol. 9995. Lecture Notes in Computer Science. 2016, pp. 164–181.

[C17] Jakub Daniel, Alessandro Cimatti, Alberto Griggio, Stefano Tonetta, and Sergio Mover. "Infinite-State Liveness-to-Safety via Implicit Abstraction and Well-Founded Relations". In: *CAV (1)*. Vol. 9779. Lecture Notes in Computer Science. Springer, 2016, pp. 271–291.

[C18] Alessandro Cimatti, Sergio Mover, and Mirko Sessa. "SMT-based analysis of switching multi-domain linear Kirchhoff networks". In: *FMCAD*. IEEE, 2017, pp. 188–195.

[C19] Roberto Cavada, Alessandro Cimatti, Sergio Mover, Mirko Sessa, Giuseppe Cadavero, and Giuseppe Scaglione. "Analysis of Relay Interlocking Systems via SMT-based Model Checking of Switched Multi-Domain Kirchhoff Networks". In: *FMCAD*. IEEE, 2018, pp. 1–9.

[C20] Sergio Mover, Sriram Sankaranarayanan, Rhys Braginton Pettee Olsen, and Bor-Yuh Evan Chang. "Mining framework usage graphs from app corpora". In: *SANER*. IEEE Computer Society, 2018, pp. 277–289.

[C21] Arjun Radhakrishna, Nicholas V. Lewchenko, Shawn Meier, Sergio Mover, Krishna Chaitanya Sripada, Damien Zufferey, Bor-Yuh Evan Chang, and Pavol Cerný. "DroidStar: callback typestates for Android classes". In: *ICSE*. ACM, 2018, pp. 1160–1170.

[C22] Goran Frehse, Alessandro Abate, Dieky Adzkiya, Anna Becchi, Lei Bu, Alessandro Cimatti, Mirco Giacobbe, Alberto Griggio, Sergio Mover, Muhammad Syifa'ul Mufid, Idriss Riouak, Stefano Tonetta, and Enea Zaffanella. "ARCH-COMP19 Category Report: Hybrid Systems with Piecewise Constant Dynamics". In: *ARCH@CPSIoTWeek*. Vol. 61. EPiC Series in Computing. EasyChair, 2019, pp. 1–13.

[C23] Shawn Meier, Sergio Mover, and Bor-Yuh Evan Chang. "Lifestate: Event-Driven Protocols and Callback Control Flow". In: *ECOOP*. Vol. 134. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 1:1–1:29.

[C24] Sriram Sankaranarayanan, Souradeep Dutta, and Sergio Mover. "Reaching Out Towards Fully Verified Autonomous Systems". In: *RP*. Vol. 11674. Lecture Notes in Computer Science. Springer, 2019, pp. 22–32.

[C25] Sergio Mover, Alessandro Cimatti, Alberto Griggio, Ahmed Irfan, and Stefano Tonetta. "Implicit Semi-Algebraic Abstraction for Polynomial Dynamical Systems". In: *CAV (1)*. Vol. 12759. Lecture Notes in Computer Science. Springer, 2021, pp. 529–551.

[C26] Stylianos Basagiannis, Ludovico Battista, Anna Becchi, Alessandro Cimatti, Georgios Giantamidis, Sergio Mover, Alberto Tacchella, Stefano Tonetta, and Vassilios A. Tsachouridis. "SMT-Based Stability Verification of an Industrial Switched PI Control Systems". In: *DSN-W*. IEEE, 2023, pp. 243–250.

[C27] Mehdi Zadem, Sergio Mover, and Sao Mai Nguyen. "Goal Space Abstraction in Hierarchical Reinforcement Learning via Set-Based Reachability Analysis". In: *ICDL*. IEEE, 2023, pp. 0–0.

[C28] Mehdi Zadem, Sergio Mover, and Sao Mai Nguyen. "Reconciling Spatial and Temporal Abstractions for Goal Representation". In: *ICLR*. OpenReview.net, 2024.

Journal Articles

[J1] Marco Bozzano, Alessandro Cimatti, Oleg Lisagor, Cristian Mattarei, Sergio Mover, Marco Roveri, and Stefano Tonetta. "Symbolic Model Checking and Safety Assessment of Altarica models". In: *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.* 46 (2011).

[J2] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. "SMT-based scenario verification for hybrid systems". In: *Formal Methods Syst. Des.* 42.1 (2013), pp. 46–66.

[J3] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. "Quantifier-free encoding of invariants for hybrid systems". In: *Formal Methods Syst. Des.* 45.2 (2014), pp. 165–188.

[J4] Marco Bozzano, Alessandro Cimatti, Oleg Lisagor, Cristian Mattarei, Sergio Mover, Marco Roveri, and Stefano Tonetta. "Safety assessment of AltaRica models via symbolic model checking". In: *Sci. Comput. Program.* 98 (2015), pp. 464–483.

[J5] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. "Infinite-state invariant checking with IC3 and predicate abstraction". In: *Formal Methods Syst. Des.* 49.3 (2016), pp. 190–218.

[J6] Xin Chen, Sergio Mover, and Sriram Sankaranarayanan. "Compositional Relational Abstraction for Nonlinear Hybrid Systems". In: *ACM Trans. Embed. Comput. Syst.* 16.5s (2017), 187:1–187:19.

[J7] Alessandro Cimatti, Alberto Griggio, Sergio Mover, Marco Roveri, and Stefano Tonetta. "Verification Modulo Theories". In: *Formal Methods Syst. Des.* 60.3 (2022), pp. 452–481.

[J8] Shawn Meier, Sergio Mover, Gowtham Kaki, and Bor-Yuh Evan Chang. "Historia: Refuting Callback Reachability with Message-History Logics". In: *Proc. ACM Program. Lang.* 7.OOPSLA2 (2023), pp. 1905–1934.

Phd Thesis

[T1] Sergio Mover. "Verification of Hybrid Systems using Satisfiability Modulo Theories". University of Trento, Italy, 2014.

Bibliography

[1] Mitio Nagumo. "Über die lage der integralkurven gewöhnlicher differentialgleichungen". In: *Proceedings of the Physico-Mathematical Society of Japan. 3rd Series* 24 (1942), pp. 551–559.

[2] G. S. Tseitin. "On the complexity of derivation in propositional calculus". In: *Studies in Constructive Mathematics and Mathematical Logic, Part 2* (1968), pp. 115–125.

[3] Patrick Cousot and Radhia Cousot. "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints". In: *POPL*. ACM, 1977, pp. 238–252.

[4] Rajeev Alur and David L. Dill. "A Theory of Timed Automata". In: *Theor. Comput. Sci.* 126.2 (1994), pp. 183–235.

[5] Edmund M. Clarke, Orna Grumberg, and David E. Long. "Model Checking and Abstraction". In: *ACM Trans. Program. Lang. Syst.* 16.5 (1994), pp. 1512–1542.

[6] Zohar Manna and Amir Pnueli. *Temporal verification of reactive systems - safety*. Springer, 1995.

[7] Moshe Y. Vardi. "An Automata-Theoretic Approach to Linear Temporal Logic". In: *Banff Higher Order Workshop*. Vol. 1043. Lecture Notes in Computer Science. Springer, 1995, pp. 238–266.

[8] Thomas A. Henzinger. "The Theory of Hybrid Automata". In: *LICS*. IEEE Computer Society, 1996, pp. 278–292.

[9] Susanne Graf and Hassen Saïdi. "Construction of Abstract State Graphs with PVS". In: *CAV*. Vol. 1254. Lecture Notes in Computer Science. Springer, 1997, pp. 72–83.

[10] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. "Algorithmic analysis of nonlinear hybrid systems". In: *IEEE Trans. Autom. Control.* 43.4 (1998), pp. 540–554.

[11] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998.

[12] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. "Symbolic Model Checking without BDDs". In: *TACAS*. Vol. 1579. Lecture Notes in Computer Science. Springer, 1999, pp. 193–207.

[13] Franco Blanchini. "Set invariance in control". In: *Automatica* 35.11 (1999), pp. 1747–1767.

[14] Stavros Tripakis. "Verifying Progress in Timed Systems". In: *ARTS*. Vol. 1601. Lecture Notes in Computer Science. Springer, 1999, pp. 299–314.

[15] Rajeev Alur, Thomas A. Henzinger, Gerardo Lafferriere, and George J. Pappas. "Discrete abstractions of hybrid systems". In: *Proc. IEEE* 88.7 (2000), pp. 971–984.

[16] Franck Cassez and Kim Guldstrand Larsen. "The Impressive Power of Stopwatches". In: *CONCUR*. Vol. 1877. Lecture Notes in Computer Science. Springer, 2000, pp. 138–152.

[17] Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. "Checking Safety Properties Using Induction and a SAT-Solver". In: *FMCAD*. Vol. 1954. Lecture Notes in Computer Science. Springer, 2000, pp. 108–125.

[18] Thomas Ball, Rupak Majumdar, Todd D. Millstein, and Sriram K. Rajamani. “Automatic Predicate Abstraction of C Programs”. In: *PLDI*. ACM, 2001, pp. 203–213.

[19] A. Goriely. “Integrability and Nonintegrability of Dynamical Systems”. In: 2001.

[20] Armin Biere, Cyrille Artho, and Viktor Schuppan. “Liveness Checking as Safety Checking”. In: *FMICS*. Vol. 66. Electronic Notes in Theoretical Computer Science 2. Elsevier, 2002, pp. 160–177.

[21] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Grégoire Sutre. “Lazy abstraction”. In: *POPL*. ACM, 2002, pp. 58–70.

[22] Thomas Ball, Andreas Podelski, and Sriram K. Rajamani. “Boolean and Cartesian abstraction for model checking C programs”. In: *Int. J. Softw. Tools Technol. Transf.* 5.1 (2003), pp. 49–58.

[23] E. M. Clarke, A. Fehnker, Z. Han, B. H. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald. “Abstraction and Counterexample-Guided Refinement in Model Checking of Hybrid Systems”. In: *Int. J. Found. Comput. Sci.* 14.4 (2003), pp. 583–604.

[24] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. “Counterexample-guided abstraction refinement for symbolic model checking”. In: *J. ACM* 50.5 (2003), pp. 752–794.

[25] Kenneth L. McMillan. “Interpolation and SAT-Based Model Checking”. In: *CAV*. Vol. 2725. Lecture Notes in Computer Science. Springer, 2003, pp. 1–13.

[26] Leonardo Mendonça de Moura, Harald Rueß, and Maria Sorea. “Bounded Model Checking and Induction: From Refutation to Verification (Extended Abstract, Category A)”. In: *CAV*. Vol. 2725. Lecture Notes in Computer Science. Springer, 2003, pp. 14–26.

[27] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth L. McMillan. “Abstractions from proofs”. In: *POPL*. ACM, 2004, pp. 232–244.

[28] Oded Maler and Dejan Nickovic. “Monitoring Temporal Properties of Continuous Signals”. In: *FORMATS/FTRTFT*. Vol. 3253. Lecture Notes in Computer Science. Springer, 2004, pp. 152–166.

[29] Ittai Balaban, Amir Pnueli, and Lenore D. Zuck. “Ranking Abstraction as Companion to Predicate Abstraction”. In: *FORTE*. Vol. 3731. Lecture Notes in Computer Science. Springer, 2005, pp. 1–12.

[30] Viktor Schuppan and Armin Biere. “Liveness Checking as Safety Checking for Infinite State Spaces”. In: *INFINITY*. Vol. 149. Electronic Notes in Theoretical Computer Science 1. Elsevier, 2005, pp. 79–96.

[31] Rajeev Alur, Thao Dang, and Franjo Ivancic. “Predicate abstraction for reachability analysis of hybrid systems”. In: *ACM Trans. Embed. Comput. Syst.* 5.1 (2006), pp. 152–199.

[32] Ranjit Jhala and Kenneth L. McMillan. “A Practical and Complete Approach to Predicate Refinement”. In: *TACAS*. Vol. 3920. Lecture Notes in Computer Science. Springer, 2006, pp. 459–473.

[33] Shuvendu K. Lahiri, Robert Nieuwenhuis, and Albert Oliveras. “SMT Techniques for Fast Predicate Abstraction”. In: *CAV*. Vol. 4144. Lecture Notes in Computer Science. Springer, 2006, pp. 424–437.

[34] Kenneth L. McMillan. “Lazy Abstraction with Interpolants”. In: *CAV*. Vol. 4144. Lecture Notes in Computer Science. Springer, 2006, pp. 123–136.

[35] Stephen Prajna. “Barrier certificates for nonlinear model validation”. In: *Autom.* 42.1 (2006), pp. 117–126.

[36] Aaron R. Bradley and Zohar Manna. *The calculus of computation - decision procedures with applications to verification*. Springer, 2007.

[37] Joël Ouaknine and James Worrell. "On the decidability and complexity of Metric Temporal Logic over finite words". In: *Log. Methods Comput. Sci.* 3.1 (2007).

[38] Andreas Podelski and Andrey Rybalchenko. "Transition predicate abstraction and fair termination". In: *ACM Trans. Program. Lang. Syst.* 29.3 (2007), p. 15.

[39] Andreas Podelski and Silke Wagner. "Region Stability Proofs for Hybrid Systems". In: *FORMATS*. Vol. 4763. Lecture Notes in Computer Science. Springer, 2007, pp. 320–335.

[40] Goran Frehse. "PHAVer: algorithmic verification of hybrid systems past HyTech". In: *Int. J. Softw. Tools Technol. Transf.* 10.3 (2008), pp. 263–279.

[41] Christian Herde, Andreas Eggers, Martin Fränzle, and Tino Teige. "Analysis of Hybrid Systems Using HySAT". In: *ICONS*. IEEE Computer Society, 2008, pp. 196–201.

[42] André Platzer. "Differential Dynamic Logic for Hybrid Systems". In: *J. Autom. Reason.* 41.2 (2008), pp. 143–189.

[43] Ashish Tiwari. "Abstractions for hybrid systems". In: *Formal Methods Syst. Des.* 32.1 (2008), pp. 57–83.

[44] Stefano Tonetta. "Abstract Model Checking without Computing the Abstraction". In: *FM*. Vol. 5850. Lecture Notes in Computer Science. Springer, 2009, pp. 89–105.

[45] Yakir Vizel and Orna Grumberg. "Interpolation-sequence based model checking". In: *FMCAD*. IEEE, 2009, pp. 1–8.

[46] Mohamed H. Zaki, William Denman, Sofiène Tahar, and Guy Bois. "Integrating Abstraction Techniques for Formal Verification of Analog Designs". In: *J. Aerosp. Comput. Inf. Commun.* 6.5 (2009), pp. 373–392.

[47] Michael R. Clarkson and Fred B. Schneider. "Hyperproperties". In: *J. Comput. Secur.* 18.6 (2010), pp. 1157–1210.

[48] Thao Dang, Oded Maler, and Romain Testylier. "Accurate hybridization of nonlinear systems". In: *HSCC*. ACM, 2010, pp. 11–20.

[49] Alexandre Donzé. "Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems". In: *CAV*. Vol. 6174. Lecture Notes in Computer Science. Springer, 2010, pp. 167–170.

[50] David Monniaux. "Quantifier Elimination by Lazy Model Enumeration". In: *CAV*. Vol. 6174. Lecture Notes in Computer Science. Springer, 2010, pp. 585–599.

[51] Rajeev Alur. "Formal verification of hybrid systems". In: *EMSOFT*. ACM, 2011, pp. 273–278.

[52] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. "S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems". In: *TACAS*. Vol. 6605. Lecture Notes in Computer Science. Springer, 2011, pp. 254–257.

[53] Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. "Secure information flow by self-composition". In: *Math. Struct. Comput. Sci.* 21.6 (2011), pp. 1207–1252.

[54] Ananda Basu, Saddek Bensalem, Marius Bozga, Jacques Combaz, Mohamad Jaber, Thanh-Hung Nguyen, and Joseph Sifakis. "Rigorous Component-Based System Design Using the BIP Framework". In: *IEEE Softw.* 28.3 (2011), pp. 41–48.

[55] Aaron R. Bradley. "SAT-Based Model Checking without Unrolling". In: *VMCAI*. Vol. 6538. Lecture Notes in Computer Science. Springer, 2011, pp. 70–87.

[56] Aaron R. Bradley, Fabio Somenzi, Zyad Hassan, and Yan Zhang. "An incremental approach to model checking progress properties". In: *FMCAD*. FMCAD Inc., 2011, pp. 144–153.

[57] Niklas Eén, Alan Mishchenko, and Robert K. Brayton. "Efficient implementation of property directed reachability". In: *FMCAD*. FMCAD Inc., 2011, pp. 125–134.

[58] Ashutosh Gupta, Corneliu Popeea, and Andrey Rybalchenko. “Predicate abstraction and refinement for verifying multi-threaded programs”. In: *POPL*. ACM, 2011, pp. 331–344.

[59] Temesghen Kahsai, Yeting Ge, and Cesare Tinelli. “Instantiation-Based Invariant Discovery”. In: *NASA Formal Methods*. Vol. 6617. Lecture Notes in Computer Science. Springer, 2011, pp. 192–206.

[60] Jiang Liu, Naijun Zhan, and Hengjun Zhao. “Computing semi-algebraic invariants for polynomial dynamical systems”. In: *EMSOFT*. ACM, 2011, pp. 97–106.

[61] Andreas Podelski and Andrey Rybalchenko. “Transition Invariants and Transition Predicate Abstraction for Program Termination”. In: *TACAS*. Vol. 6605. Lecture Notes in Computer Science. Springer, 2011, pp. 3–10.

[62] Sriram Sankaranarayanan and Ashish Tiwari. “Relational Abstractions for Continuous and Hybrid Systems”. In: *CAV*. Vol. 6806. Lecture Notes in Computer Science. Springer, 2011, pp. 686–702.

[63] Aws Albarghouthi, Arie Gurfinkel, and Marsha Chechik. “From Under-Approximations to Over-Approximations and Back”. In: *TACAS*. Vol. 7214. Lecture Notes in Computer Science. Springer, 2012, pp. 157–172.

[64] Alessandro Cimatti and Alberto Griggio. “Software Model Checking via IC3”. In: *CAV*. Vol. 7358. Lecture Notes in Computer Science. Springer, 2012, pp. 277–293.

[65] Koen Claessen and Niklas Sörensson. “A liveness checking algorithm that counts”. In: *FMCAD*. IEEE, 2012, pp. 52–59.

[66] William Denman. “Abstracting Continuous Nonpolynomial Dynamical Systems”. In: *ICCSW*. Vol. 28. OASIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2012, pp. 42–48.

[67] Krystof Hoder and Nikolaj S. Bjørner. “Generalized Property Directed Reachability”. In: *SAT*. Vol. 7317. Lecture Notes in Computer Science. Springer, 2012, pp. 157–171.

[68] Roland Kindermann, Tommi A. Junttila, and Ilkka Niemelä. “SMT-Based Induction Methods for Timed Systems”. In: *FORMATS*. Vol. 7595. Lecture Notes in Computer Science. Springer, 2012, pp. 171–187.

[69] Ashish Tiwari. “HybridSAL Relational Abstracter”. In: *CAV*. Vol. 7358. Lecture Notes in Computer Science. Springer, 2012, pp. 725–731.

[70] Emanuel Todorov, Tom Erez, and Yuval Tassa. “MuJoCo: A physics engine for model-based control”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 5026–5033.

[71] Aditya Zutshi, Sriram Sankaranarayanan, and Ashish Tiwari. “Timed Relational Abstractions for Sampled Data Control Systems”. In: *CAV*. Vol. 7358. Lecture Notes in Computer Science. Springer, 2012, pp. 343–361.

[72] Rajeev Alur, Rastislav Bodík, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. “Syntax-guided synthesis”. In: *FMCAD*. IEEE, 2013, pp. 1–8.

[73] Davide Bresolin. “Improving HyLTL model checking of hybrid systems”. In: *GandALF*. Vol. 119. EPTCS. 2013, pp. 79–92.

[74] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. “Flow*: An Analyzer for Non-linear Hybrid Systems”. In: *CAV*. Vol. 8044. Lecture Notes in Computer Science. Springer, 2013, pp. 258–263.

[75] Sicun Gao, Soonho Kong, and Edmund M. Clarke. “dReal: An SMT Solver for Nonlinear Theories over the Reals”. In: *CADE*. Vol. 7898. Lecture Notes in Computer Science. Springer, 2013, pp. 208–214.

[76] Matthias Heizmann, Jochen Hoenicke, Jan Leike, and Andreas Podelski. “Linear Ranking for Linear Lasso Programs”. In: *ATVA*. Vol. 8172. Lecture Notes in Computer Science. Springer, 2013, pp. 365–380.

[77] Hui Kong, Fei He, Xiaoyu Song, William N. N. Hung, and Ming Gu. “Exponential-Condition-Based Barrier Certificate Generation for Safety Verification of Hybrid Systems”. In: *CAV*. 2013, pp. 242–257.

[78] Tobias Welp and Andreas Kuehlmann. “QF BV model checking with property directed reachability”. In: *DATE*. EDA Consortium San Jose, CA, USA / ACM DL, 2013, pp. 791–796.

[79] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick D. McDaniel. “FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps”. In: *PLDI*. ACM, 2014, pp. 259–269.

[80] Johannes Birgmeier, Aaron R. Bradley, and Georg Weissenbacher. “Counterexample to Induction-Guided Abstraction-Refinement (CTIGAR)”. In: *CAV*. Vol. 8559. Lecture Notes in Computer Science. Springer, 2014, pp. 831–848.

[81] Bor-Yuh Evan Chang. “Refuting Heap Reachability”. In: *VMCAI*. Vol. 8318. Lecture Notes in Computer Science. Springer, 2014, pp. 137–141.

[82] Eric Goubault, Jacques-Henri Jourdan, Sylvie Putot, and Sriram Sankaranarayanan. “Finding non-polynomial positive invariants and lyapunov functions for polynomial systems through Darboux polynomials”. In: *ACC*. 2014, pp. 3571–3578.

[83] Shachar Itzhaky, Nikolaj S. Bjørner, Thomas W. Reps, Mooly Sagiv, and Aditya V. Thakur. “Property-Directed Shape Analysis”. In: *CAV*. Vol. 8559. Lecture Notes in Computer Science. Springer, 2014, pp. 35–51.

[84] Bertrand Jeannet, Peter Schrammel, and Sriram Sankaranarayanan. “Abstract acceleration of general linear loops”. In: *POPL*. ACM, 2014, pp. 529–540.

[85] Dirk Beyer, Matthias Dangl, and Philipp Wendler. “Boosting k-Induction with Continuously-Refined Invariants”. In: *CAV (1)*. Vol. 9206. Lecture Notes in Computer Science. Springer, 2015, pp. 622–640.

[86] Nikolaj S. Bjørner and Arie Gurfinkel. “Property Directed Polyhedral Abstraction”. In: *VMCAI*. Vol. 8931. Lecture Notes in Computer Science. Springer, 2015, pp. 263–281.

[87] Sam Blackshear, Alexandra Gendreau, and Bor-Yuh Evan Chang. “Droidel: a general approach to Android framework modeling”. In: *SOAP@PLDI*. ACM, 2015, pp. 19–25.

[88] Martin Brain, Saurabh Joshi, Daniel Kroening, and Peter Schrammel. “Safety Verification and Refutation by k-Invariants and k-Induction”. In: *SAS*. Vol. 9291. Lecture Notes in Computer Science. Springer, 2015, pp. 145–161.

[89] Xin Chen. “Reachability analysis of non-linear hybrid systems using Taylor Models”. PhD thesis. RWTH Aachen University, Germany, 2015.

[90] Alessandro Cimatti, Marco Roveri, and Stefano Tonetta. “HRELTL: A temporal logic for hybrid systems”. In: *Inf. Comput.* 245 (2015), pp. 54–71.

[91] William Denman. “Automated verification of continuous and hybrid dynamical systems”. PhD thesis. University of Cambridge, UK, 2015.

[92] Daniel Dietsch, Matthias Heizmann, Vincent Langenfeld, and Andreas Podelski. “Fairness Modulo Theory: A New Approach to LTL Software Model Checking”. In: *CAV (1)*. Vol. 9206. Lecture Notes in Computer Science. Springer, 2015, pp. 49–66.

[93] Nathan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völp, and André Platzer. “KeYmaera X: An Axiomatic Tactical Theorem Prover for Hybrid Systems”. In: *CADE*. 2015, pp. 527–538.

[94] Aleksandr Karbyshev, Nikolaj S. Bjørner, Shachar Itzhaky, Noam Rinetzky, and Sharon Shoham. "Property-Directed Inference of Universal Invariants or Proving Their Absence". In: *CAV (1)*. Vol. 9206. Lecture Notes in Computer Science. Springer, 2015, pp. 583–602.

[95] Anvesh Komuravelli, Nikolaj S. Bjørner, Arie Gurfinkel, and Kenneth L. McMillan. "Compositional Verification of Procedural Programs using Horn Clauses over Integers and Arrays". In: *FMCAD*. IEEE, 2015, pp. 89–96.

[96] Ashish Tiwari. "Time-Aware Abstractions in HybridSal". In: *CAV (1)*. Vol. 9206. Lecture Notes in Computer Science. Springer, 2015, pp. 504–510.

[97] Shuling Wang, Naijun Zhan, and Liang Zou. "An Improved HHL Prover: An Interactive Theorem Prover for Hybrid Systems". In: *ICFEM*. 2015, pp. 382–399.

[98] Aws Albarghouthi, Isil Dillig, and Arie Gurfinkel. "Maximal specification synthesis". In: *POPL*. ACM, 2016, pp. 789–801.

[99] Benjamin Bittner, Marco Bozzano, Roberto Cavada, Alessandro Cimatti, Marco Gario, Alberto Griggio, Cristian Mattarei, Andrea Micheli, and Gianni Zampedri. "The xSAP Safety Analysis Platform". In: *TACAS*. Vol. 9636. Lecture Notes in Computer Science. Springer, 2016, pp. 533–539.

[100] Alberto Griggio and Marco Roveri. "Comparing Different Variants of the ic3 Algorithm for Hardware Model Checking". In: *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 35.6 (2016), pp. 1026–1039.

[101] Hui Kong, Ezio Bartocci, Sergiy Bogomolov, Radu Grosu, Thomas A. Henzinger, Yu Jiang, and Christian Schilling. "Discrete Abstraction of Multiaffine Systems". In: *HSB*. Vol. 9957. Lecture Notes in Computer Science. 2016, pp. 128–144.

[102] Andrew Sogokon, Khalil Ghorbal, Paul B. Jackson, and André Platzer. "A Method for Invariant Generation for Polynomial Continuous Systems". In: *VMCAI*. Vol. 9583. Lecture Notes in Computer Science. Springer, 2016, pp. 268–288.

[103] Sergiy Bogomolov, Goran Frehse, Mirco Giacobbe, and Thomas A. Henzinger. "Counterexample-Guided Refinement of Template Polyhedra". In: *TACAS (1)*. Vol. 10205. Lecture Notes in Computer Science. 2017, pp. 589–606.

[104] Xin Chen and Sriram Sankaranarayanan. "Model Predictive Real-Time Monitoring of Linear Systems". In: *RTSS*. IEEE Computer Society, 2017, pp. 297–306.

[105] Alessandro Cimatti, Alberto Griggio, Ahmed Irfan, Marco Roveri, and Roberto Sebastiani. "Invariant Checking of NRA Transition Systems via Incremental Reduction to LRA with EUF". In: *TACAS (1)*. Vol. 10205. Lecture Notes in Computer Science. 2017, pp. 58–75.

[106] Khalil Ghorbal, Andrew Sogokon, and André Platzer. "A hierarchy of proof rules for checking positive invariance of algebraic and semi-algebraic sets". In: *Comput. Lang. Syst. Struct.* 47 (2017), pp. 19–43.

[107] Luan Viet Nguyen, James Kapinski, Xiaoqing Jin, Jyotirmoy V. Deshmukh, and Taylor T. Johnson. "Hyperproperties of real-valued signals". In: *MEMOCODE*. ACM, 2017, pp. 104–113.

[108] Nima Roohi, Pavithra Prabhakar, and Mahesh Viswanathan. "HARE: A Hybrid Abstraction Refinement Engine for Verifying Non-linear Hybrid Automata". In: *TACAS*. 2017, pp. 573–588.

[109] Stanley Bak. "t-Barrier Certificates: A Continuous Analogy to k-Induction". In: *ADHS*. Vol. 51. IFAC-PapersOnLine 16. Elsevier, 2018, pp. 145–150.

[110] Clark W. Barrett and Cesare Tinelli. "Satisfiability Modulo Theories". In: *Handbook of Model Checking*. Springer, 2018, pp. 305–343.

[111] Rose Bohrer and André Platzer. "A Hybrid, Dynamic Logic for Hybrid-Dynamic Information Flow". In: *LICS*. ACM, 2018, pp. 115–124.

[112] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, eds. *Handbook of Model Checking*. Springer, 2018.

[113] Laurent Doyen, Goran Frehse, George J. Pappas, and André Platzer. "Verification of Hybrid Systems". In: *Handbook of Model Checking*. Springer, 2018, pp. 1047–1110.

[114] Goran Frehse, Nikolaos Kekatos, Dejan Nickovic, Jens Oehlerking, Simone Schuler, Alexander Walsch, and Matthias Woehrle. "A Toolchain for Verifying Safety Properties of Hybrid Automata via Pattern Templates". In: *ACC*. IEEE, 2018, pp. 2384–2391.

[115] Andrew Gacek, John Backes, Mike Whalen, Lucas G. Wagner, and Elaheh Ghassabani. "The JKind Model Checker". In: *CAV (2)*. Vol. 10982. Lecture Notes in Computer Science. Springer, 2018, pp. 20–27.

[116] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. "Data-Efficient Hierarchical Reinforcement Learning". In: *NeurIPS*. 2018, pp. 3307–3317.

[117] André Platzer. *Logical Foundations of Cyber-Physical Systems*. Springer, 2018.

[118] Kyungmin Bae and Jia Lee. "Bounded model checking of signal temporal logic properties using syntactic separation". In: *Proc. ACM Program. Lang.* 3.POPL (2019), 51:1–51:30.

[119] Anna Becchi and Enea Zaffanella. "Revisiting Polyhedral Analysis for Hybrid Systems". In: *SAS*. Vol. 11822. Lecture Notes in Computer Science. Springer, 2019, pp. 183–202.

[120] Nikola Benes, Lubos Brim, Jana Drazanová, Samuel Pastva, and David Safránek. "Facetal abstraction for non-linear dynamical systems based on δ-decidable SMT". In: *HSCC*. ACM, 2019, pp. 99–108.

[121] Marco Bozzano, Harold Bruintjes, Alessandro Cimatti, Joost-Pieter Katoen, Thomas Noll, and Stefano Tonetta. "COMPASS 3.0". In: *TACAS (1)*. Vol. 11427. Lecture Notes in Computer Science. Springer, 2019, pp. 379–385.

[122] Alessandro Cimatti, Alberto Griggio, Enrico Magnago, Marco Roveri, and Stefano Tonetta. "Extending nuXmv with Timed Transition Systems and Timed Temporal Properties". In: *CAV (1)*. Vol. 11561. Lecture Notes in Computer Science. Springer, 2019, pp. 376–386.

[123] Jyotirmoy Deshmukh and Sriram Sankaranarayanan. "Formal Techniques for Verification and Testing of Cyber-Physical Systems". In: *Design Automation of Cyber-Physical Systems (Edited by Arquimedes Canedo and Mohammad Al Faruque)*. Springer-Verlag, 2019, pp. 69–105.

[124] Aman Goel and Karem A. Sakallah. "Model Checking of Verilog RTL Using IC3 with Syntax-Guided Abstraction". In: *NFM*. Vol. 11460. Lecture Notes in Computer Science. Springer, 2019, pp. 166–185.

[125] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. "Near-Optimal Representation Learning for Hierarchical Reinforcement Learning". In: *ICLR (Poster)*. OpenReview.net, 2019.

[126] Luan Viet Nguyen, Gautam Mohan, James Weimer, Oleg Sokolsky, Insup Lee, and Rajeev Alur. "Detecting security leaks in hybrid systems with information flow analysis". In: *MEMOCODE*. ACM, 2019, 14:1–14:11.

[127] Ron Shemer, Arie Gurfinkel, Sharon Shoham, and Yakir Vizel. "Property Directed Self Composition". In: *CAV (1)*. Vol. 11561. Lecture Notes in Computer Science. Springer, 2019, pp. 161–179.

[128] Andrew Sogokon, Paul B. Jackson, and Taylor T. Johnson. "Verifying Safety and Persistence in Hybrid Systems Using Flowpipes and Continuous Invariants". In: *J. Autom. Reason.* 63.4 (2019), pp. 1005–1029.

[129] Andrew Sogokon, Stefan Mitsch, Yong Kiam Tan, Katherine Cordwell, and André Platzer. "Pegasus: A Framework for Sound Continuous Invariant Generation". In: *FM*. 2019, pp. 138–157.

[130] Alessandro Cimatti, Alberto Griggio, Enrico Magnago, Marco Roveri, and Stefano Tonetta. "SMT-based satisfiability of first-order LTL with event freezing functions and metric operators". In: *Inf. Comput.* 272 (2020), p. 104502.

[131] Hari Govind V. K., Grigory Fedyukovich, and Arie Gurfinkel. "Word Level Property Directed Reachability". In: *ICCAD*. IEEE, 2020, 107:1–107:9.

[132] Philipp Rümmer. "Competition Report: CHC-COMP-20". In: *arXiv preprint arXiv:2008.02939* (2020).

[133] Philipp Rümmer. "Competition Report: CHC-COMP-20". In: *Electronic Proceedings in Theoretical Computer Science* 320 (Aug. 2020), 197–219. ISSN: 2075-2180. URL: http://dx.doi.org/10.4204/EPTCS.320.15.

[134] Hammad Ahmad and Jean-Baptiste Jeannin. "A program logic to verify signal temporal logic specifications of hybrid systems". In: *HSCC*. ACM, 2021, 10:1–10:11.

[135] Matthias Althoff, Goran Frehse, and Antoine Girard. "Set Propagation Techniques for Reachability Analysis". In: *Annu. Rev. Control. Robotics Auton. Syst.* 4 (2021), pp. 369–395.

[136] Clark Barrett, Roberto Sebastiani, Sanjit Seshia, and Cesare Tinelli. "Satisfiability Modulo Theories". In: *Handbook of Satisfiability, Second Edition*. Ed. by Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh. Vol. 336. Frontiers in Artificial Intelligence and Applications. IOS Press, Feb. 2021. Chap. 33, pp. 825–885. URL: http://theory.stanford.edu/~barrett/pubs/BSST21.pdf.

[137] Dejan Jovanovic and Bruno Dutertre. "Interpolation and Model Checking for Nonlinear Arithmetic". In: *CAV (2)*. Vol. 12760. Lecture Notes in Computer Science. Springer, 2021, pp. 266–288.

[138] Siyuan Li, Lulu Zheng, Jianhao Wang, and Chongjie Zhang. "Learning Subgoal Representations with Slow Dynamics". In: *ICLR*. OpenReview.net, 2021.

[139] Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher A. Strong, Clark W. Barrett, and Mykel J. Kochenderfer. "Algorithms for Verifying Deep Neural Networks". In: *Found. Trends Optim.* 4.3-4 (2021).

[140] Makai Mann, Ahmed Irfan, Florian Lonsing, Yahan Yang, Hongce Zhang, Kristopher Brown, Aarti Gupta, and Clark W. Barrett. "Pono: A Flexible and Extensible SMT-Based Model Checker". In: *CAV (2)*. Vol. 12760. Lecture Notes in Computer Science. Springer, 2021, pp. 461–474.

[141] Qiuye Wang, Mingshuai Chen, Bai Xue, Naijun Zhan, and Joost-Pieter Katoen. "Synthesizing Invariant Barrier Certificates via Difference-of-Convex Programming". In: *CAV (1)*. Vol. 12759. Lecture Notes in Computer Science. Springer, 2021, pp. 443–466.

[142] Raven Beutner and Bernd Finkbeiner. "Software Verification of Hyperproperties Beyond k-Safety". In: *CAV (1)*. Vol. 13371. Lecture Notes in Computer Science. Springer, 2022, pp. 341–362.

[143] Dirk Beyer, Nian-Ze Lee, and Philipp Wendler. "Interpolation and SAT-Based Model Checking Revisited: Adoption to Software Verification". In: *CoRR* abs/2208.05046 (2022).

[144] Martin Blicha, Grigory Fedyukovich, Antti E. J. Hyvärinen, and Natasha Sharygina. "Split Transition Power Abstraction for Unbounded Safety". In: *FMCAD*. IEEE, 2022, pp. 349–358.

[145] Goran Frehse, Mirco Giacobbe, and Enea Zaffanella. "Symbolic Analysis of Linear Hybrid Automata - 25 Years Later". In: *Principles of Systems Design*. Vol. 13660. Lecture Notes in Computer Science. Springer, 2022, pp. 39–60.

[146] Khalil Ghorbal and Andrew Sogokon. "Characterizing positively invariant sets: Inductive and topological methods". In: *J. Symb. Comput.* 113 (2022), pp. 1–28.

[147] Hari Govind V. K., Sharon Shoham, and Arie Gurfinkel. “Solving constrained Horn clauses modulo algebraic data types and recursive functions”. In: *Proc. ACM Program. Lang.* 6.POPL (2022), pp. 1–29.

[148] Makai Mann, Ahmed Irfan, Alberto Griggio, Oded Padon, and Clark W. Barrett. “Counterexample-Guided Prophecy for Model Checking Modulo the Theory of Arrays”. In: *Log. Methods Comput. Sci.* 18.3 (2022).

[149] Shubham Pateria, Budhitama Subagdja, Ah-Hwee Tan, and Chai Quek. “Hierarchical Reinforcement Learning: A Comprehensive Survey”. In: *ACM Comput. Surv.* 54.5 (2022), 109:1–109:35.

[150] Bernd Finkbeiner, Hadar Frenkel, Jana Hofmann, and Janine Lohse. “Automata-Based Software Model Checking of Hyperproperties”. In: *NFM*. Vol. 13903. Lecture Notes in Computer Science. Springer, 2023, pp. 361–379.

[151] Isabel Garcia-Contreras, Hari Govind V. K., Sharon Shoham, and Arie Gurfinkel. “Fast Approximations of Quantifier Elimination”. In: *CAV (2)*. Vol. 13965. Lecture Notes in Computer Science. Springer, 2023, pp. 64–86.

[152] Alberto Griggio and Martin Jonás. “Kratos2: An SMT-Based Model Checker for Imperative Programs”. In: *CAV (3)*. Vol. 13966. Lecture Notes in Computer Science. Springer, 2023, pp. 423–436.

[153] Tianren Zhang, Shangqi Guo, Tian Tan, Xiaolin Hu, and Feng Chen. “Adjacency Constraint for Efficient Hierarchical Reinforcement Learning”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 45.4 (2023), pp. 4152–4166.