

© 2017 Nima Roohi

REMEDIES FOR BUILDING RELIABLE CYBER-PHYSICAL SYSTEMS

BY

NIMA ROOHI

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

Doctoral Committee:

Professor Mahesh Viswanathan, Chair
Professor Geir E. Dullerud
Associate Professor Sayan Mitra
Assistant Professor Sasa Misailovic
Doctor Ashish Tivari, SRI International

Abstract

Cyber-physical systems (CPS) are systems that are tight integration of computer programs as controllers or cyber parts, and physical environments. The interaction is carried out by obtaining information about the physical environment through reading sensors and responding to the current knowledge through actuators. Examples of such systems are autonomous automobile systems, avionic systems, robotic systems, and medical devices. Perhaps the most common feature of all these systems is that they are all safety critical systems and failure most likely causes catastrophic consequences. This means that while testing continues to increase confidence in cyber-physical systems, formal or mathematical proofs are needed at the very least for the safety requirements of these systems.

Hybrid automata is the main modeling language for cyber-physical systems. However, verifying safety properties is undecidable for all but very restricted known classes of these automata. Our first result introduces a new subclass of hybrid automata for which bounded time safety model checking problem is decidable. We also prove that unbounded time model checking for this subclass is undecidable which suggests this is the best one can hope for the new class. Our second result in this thesis is a counter-example guided abstraction refinement algorithm for unbounded time model checking of non-linear hybrid automata. Clearly, this is an undecidable problem and that is the main reason for using abstraction refinement techniques. Our CEGAR framework for this class is sound but not complete, meaning the algorithm never incorrectly says a system is safe, but may output unsafe incorrectly. We have also implemented our algorithm and compared it with seven other tools.

There are multiple inherent problems with traditional model checking approaches. First, it is well-known that most models do not depict physical environments precisely. Second, the model checking problem is undecidable

for most classes of hybrid automata. And third, even when model checking is decidable, controller part in most models cannot be implemented. These problems suggest that current methods of modeling cyber-physical systems and problems might not be the right ones. Our last result focuses on *robust* model checking of cyber-physical systems. In this part of the thesis, we focus on the implementability issue and show how to solve four different robust model checking problem for timed automata. We also introduce an optimal algorithm for robust time bounded safety model checking of monotonic rectangular automata.

To my family.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my advisor Prof. Mahesh Viswanathan for his continuous support of my study and research during the past five years, and for his patience, understanding, and remarkable knowledge. He was always there to help me every step of the way, from discussion on courses, writing papers, and discussing proofs, to job hunting. Having such great advisors during my academic life is one the very few area in my life in which I truly feel lucky.

I would also like to thank Prof. Geir Dullerud, Prof. Sayan Mitra, and Prof. Pavithra Prabhakar for their assistance and guidance in my research. It is wonderful opportunity to see and work with someone like Prof. Dullerud that is not directly in your field and have a glimpse of how such a well-known researcher tackles a problem. Prof. Mitra is the one who officially introduced me to cyber-physical systems and hybrid automata and I am grateful for that. Prof. Prabhakar is a former student of my advisor and many of my research interests are aligned with hers. It is amazing to know and work with somebody who is done it all not long before you.

I would also like to thank my parents for their love and support till this day of life. Even when they disagree with me, I never lost their support and 10000+ kilometers between us change nothing.

Next, I would like to thank my wife Leili for her unwavering love, support, encouragement, patience, and understanding during my study here. I understand that it is not always easy to be with me and her tolerance in difficult times is a testament of her undeniable devotion and love.

Finally, I would like to thank Krehbiel and Erickson families for all their helps during the time I have been in the US. They made starting a new life and its adjustments a lot easier for me and my wife here. I can only wish that there were more kind, welcoming, and helpful people like them everywhere.

Table of Contents

List of Figures	viii
List of Algorithms	ix
List of Symbols	x
Chapter 1 Introduction	1
1.1 Structure of the Dissertation	5
Chapter 2 Preliminaries	7
2.1 Sets and Functions	7
2.2 Transition Systems and Hybrid Automata	8
2.2.1 Subclasses of Hybrid Automata	11
2.2.2 Trajectories and Executions	13
2.2.3 Perturbation of Hybrid Automata	14
Chapter 3 Exact Model Checking	17
3.1 Unbounded Time Reachability	19
3.2 Translating Initialized Linear Inclusion Automata to Irrational Timed Automata	22
3.2.1 From Linear Inclusion to Linear Equation	24
3.2.2 From Linear Equation to Stopwatch	28
3.3 Bounding the Execution Length in Irrational Timed Automata	31
3.3.1 Brihaye <i>et al.</i> 's Algorithm for Monotonic Rectangular Automata	32
3.3.2 Time-Bounded Reachability for Irrational Timed Automata	34
3.3.3 Algorithm for Time-Bounded Reachability	37
3.3.4 Time-Bounded Reachability is PSPACE-hard in Initialized Linear Inclusion Automata	38
3.4 Related Work	39
3.5 Conclusions	39

Chapter 4	Approximate Model Checking	40
4.1	Computing Reachable Sets	43
4.1.1	Time-Bounded Transitions	43
4.1.2	Abstraction	45
4.1.3	Counter-example and Model Checking Rectangular Automata	47
4.1.4	Validating Abstract Counter-examples	48
4.1.5	Refinement	50
4.1.6	Validation Approximation	52
4.1.7	Need for Forward and Backward Reachability	53
4.2	Arbitrary Approximation of Reachable Set for Non-linear Dynamics	56
4.3	Architecture of HARE	63
4.4	Experimental Results	64
4.4.1	Unbounded Invariants	68
4.5	Related Work	69
4.6	Conclusions	70
Chapter 5	Robust Model Checking	72
5.1	Bounded Time Robust Reachability	75
5.1.1	From Bounded Time to Bounded Length	76
5.1.2	NEXPTIME Algorithm	77
5.1.3	NEXPTIME-hardness	80
5.2	Robust ω -Regular Model Checking	84
5.2.1	Robust ω -Regular Model Checking When Both Guards and Clocks are Perturbed	84
5.2.2	Robust ω -Regular Model Checking When Only Clocks are Perturbed	89
5.3	Computing Limit Reachable Sets under Clock Drifts	94
5.3.1	Extension to Unbounded Time Automata	101
5.4	Experimental Results	102
5.5	Related Work	103
5.6	Conclusions	105
Chapter 6	Conclusions	106
6.1	Future Directions	107
6.1.1	Improvements	107
6.1.2	New Avenues	108
References		110

List of Figures

3.1	Basic gadgets for simulating 2-counter machines	21
3.2	Gadgets that simulate 2-counter machines	22
3.3	Example Initialized Linear Inclusion Automaton \mathcal{A}	23
3.4	Example Initialized Linear Inclusion Automaton \mathcal{A}'	26
3.5	Example Initialized Linear Equation Automaton \mathcal{B}	28
3.6	Example Stopwatch Automaton \mathcal{C}	30
4.1	High Level CEGAR Loop	43
4.2	Validation and Refinement	50
4.3	Need for annotated counter-examples in refinement	55
4.4	Flow Chart of HARE's CEGAR Loop	62
5.1	Gadgets for Simulating Turing Machine	83
5.2	$\exists \epsilon : \mathbb{R}_+ \bullet \mathcal{H}^\epsilon \models B$ does not imply $\exists \delta : \mathbb{R}_+ \bullet \mathcal{H}_\delta \models B$	88
5.3	Overview of the Results in Section 5.2	94

List of Algorithms

1	High level steps of our CEGAR algorithm	44
2	Robust Reachability Analysis of Monotonic Rectangular Automata	79
3	Computing the limit reachable set of a bounded timed automaton	95

List of Symbols

$\llbracket \mathcal{H} \rrbracket_{\infty}^0$	executions of \mathcal{H} 14, 84, 85, 89
$\llbracket \mathcal{H} \rrbracket_{*}^0$	finite executions of \mathcal{H} 14, 86, 92
$\llbracket \mathcal{H} \rrbracket_{\omega}^0$	infinite executions of \mathcal{H} 14, 86, 87, 91–93
$V_{\mathcal{H}}$	exactly $\mathbb{R}^{X_{\mathcal{H}}}$ 9–12, 48, 60, 78
$D_{\mathcal{H}}(e)$	destination of edge e 10–12, 24, 47, 48, 50, 57, 59, 90
$E_{\mathcal{H}}$	edges of hybrid automaton \mathcal{H} . . . 9–15, 24, 25, 28, 29, 45, 46, 53, 84, 87, 89–93, 101
$F_{\mathcal{H}}$	flows of hybrid automaton \mathcal{H} 9–13, 15, 24–27, 29, 45, 46, 89
$G_{\mathcal{H}}(e)$	guard of edge e 10, 12, 24, 25, 89, 90
$I_{\mathcal{H}}$	invariants of hybrid automaton \mathcal{H} . 9–12, 15, 24–27, 45–47, 57, 59–62, 101
$J_{\mathcal{H}}(e)$	variables that are reset after taking e 12, 13, 24, 25, 29
$L_{\mathcal{H}}(e)$	label of edge e 10, 90, 91
$Q_{\mathcal{H}}^{\text{bad}}$	unsafe locations of hybrid automaton \mathcal{H} 46, 47
$Q_{\mathcal{H}}^{\text{init}}$	initial locations of hybrid automaton \mathcal{H} . . 10, 24, 25, 29, 46, 47, 91, 101
$Q_{\mathcal{H}}$	locations of hybrid automaton \mathcal{H} . . 9–13, 15, 24–27, 29–33, 36, 45–47, 76, 87, 89, 92, 97, 101
$R_{\mathcal{H}}^{\square}(e)$	possible values after taking edge e 12, 13
$R_{\mathcal{H}}(e)$	reset relation of edge e 10–12, 24, 25, 29, 57, 60, 89, 90, 99, 100
$S_{\mathcal{H}}(e)$	source of edge e 10–12, 24, 47, 90
$X_{\mathcal{H}}^{\text{init}}$	initial states of hybrid automaton \mathcal{H} 9–13, 15, 25, 29, 45
$X_{\mathcal{H}}$	variables of hybrid automaton \mathcal{H} . . 9–12, 15, 23–30, 33–37, 45, 46, 56, 57, 59, 60, 76, 78, 85, 87, 89, 91, 92, 95–101
$L_n(\tau)$	label of sequence τ at step n 13, 14
$\llbracket \mathcal{H} \rrbracket_{\infty}$	trajectories of \mathcal{H} 13, 92
$\llbracket \mathcal{H} \rrbracket_{*}$	finite trajectories of \mathcal{H} 13, 76, 77, 96–98
$\llbracket \mathcal{H} \rrbracket_{\omega}$	infinite trajectories of \mathcal{H} 13, 92

$\Sigma_{\mathcal{T}}$	labels of transition system \mathcal{T}	9, 10, 13
$S_{\mathcal{T}}^{\text{bad}}$	unsafe states of transition system \mathcal{T}	48
$S_{\mathcal{T}}^{\text{init}}$	initial states of transition system \mathcal{T}	9, 10, 13, 14, 76, 95
$S_{\mathcal{T}}$	states of transition system \mathcal{T}	9, 10, 13–15, 28, 30, 46, 48, 53, 90–92, 96
$[a, b]$	$\{x : \mathbb{R} \mid a \leq x \leq b\}$	8, 11–13, 15, 25, 27
$[a, b)$	$\{x : \mathbb{R} \mid a \leq x < b\}$	8, 20, 27, 88, 101
$(a, b]$	$\{x : \mathbb{R} \mid a < x \leq b\}$	8
(a, b)	$\{x : \mathbb{R} \mid a < x < b\}$	8, 25, 27
\mathbb{N}	the set of natural numbers	7, 8, 13, 20, 58, 65, 76, 81, 92, 96
\mathbb{N}_+	the set of positive natural numbers	7, 59, 60, 62, 100
\mathbb{Q}	the set of rational numbers	7, 29, 66, 77, 78
\mathbb{Q}_+	the set of positive rational numbers	7, 29, 66
\mathbb{R}	the set of real numbers	7–10, 12, 18, 25, 27, 34, 46, 56–59, 66, 78, 80, 89, 101, 104
$\mathbb{R}_{\geq 0}$	the set of non-negative real numbers	7, 10, 11, 13–15, 18, 24, 26, 58, 59, 76, 77, 80, 97, 104
\mathbb{R}_+	the set of positive real numbers	viii, 7, 15, 16, 44, 49, 53, 57–62, 76, 78, 80, 82, 84, 86–89, 93, 94, 96, 97, 103, 104
$\lceil x \rceil$	smallest integer not smaller than x	7, 97
$\lfloor x \rfloor$	largest integer not larger than x	7, 20
$\langle x \rangle$	fractional part of x	7
$\langle r \rangle$	$\frac{3^n}{2^{\lceil \lg \tau \rceil}}$	20–22
$ A $	size of A	7, 8, 10, 11, 13, 26, 30–33, 36–38, 76–78, 81, 87, 91, 95–102
$ \tau $	length of trajectory τ	13, 14, 37
$\llbracket \mathcal{H} \rrbracket$	Semantics of hybrid automaton \mathcal{H}	10, 13, 14, 28, 30, 46, 48, 53, 58–62, 76, 90–92
$\text{dec}(c)$	decrement counter c	19–22
halt	terminate the execution	19, 21, 22
$\text{inc}(c)$	increment counter c	19, 21, 22
$\text{jz}(c, \ell)$	jump to statement ℓ is $c = 0$	19, 22
\mathcal{L}	labels of a 2-Counter Machine	19
$;$	sequence operator	19
$A \times B$	Cartesian product of A and B	7, 9, 10, 13, 20, 33
$\text{cmax}_{\mathcal{H}}$	largest constant in rectangular hybrid automaton \mathcal{H}	32, 33, 76–78, 80, 97
$\tau_1 \sim \tau_2$	τ_1 concatenated with τ_2	13, 56, 57, 59, 92, 98, 99

$\text{cpost}(S)$	set of states that can be reached from S in one continuous step 49, 50, 53, 61, 63
$\text{cpre}(S)$	set of states that can reach to S in one continuous step .48, 63
$x := y$	x is by definition equal to y .5, 7–10, 13–15, 20, 45, 58–62, 78, 80–82, 85, 87, 88, 90–94, 97, 99–103
$\text{dom}(f)$	domain of function f 7, 13
$\text{dpost}(S)$	set of states that can be reached from S in one discrete step 49
$\text{dpre}(S)$	set of states that can reach to S in one discrete step 48
$\text{duration}(\tau)$	duration of trajectory τ 14, 32, 76, 77, 91, 92, 97, 98
$\text{edges}(\tau)$	edges that are visited by trajectory τ 76, 77
$\text{first}(\tau)$	first element of trajectory τ 13, 14, 76, 77, 92, 96–98
$\text{fst}(t)$	first element of tuple t 7, 13
$s_1 \rightarrow s_2$	state s_1 goes to state s_2 9–11, 13, 26, 27, 47–49, 54–57, 60, 90–92
$\text{inf}(\tau)$	locations that are visited infinitely often by τ 14, 87, 91, 92
limreach	limit of reachable set of timed automaton \mathcal{H} . . . 15, 73–75, 86, 88, 94–96, 100–102, 105
$\text{locs}(\tau)$	locations that are visited by trajectory τ 97
$\text{merge}(e_1, e_2)$	merging e_1 and e_2 90, 91
$f - g$	pointwise subtraction of function g from function f 7
$M_{\mathcal{H}}$	largest constant in timed automaton \mathcal{H} . . . 84–87, 89, 90, 93, 94, 97, 98, 101
$x : A$	x is of type A . . . viii, 5, 7–22, 24–29, 32, 34, 35, 44–46, 56–62, 76–78, 80–82, 84–101, 103, 104
$f \cap g$	pointwise intersection of function f and function g 7
$A + B$	Minkowski sum of A and B (assuming $A, B \subseteq \mathbb{R}$) 7
$f \dot{\in} g$	pointwise membership of function f in function g 7
$\mathbb{P}(\mathbf{X})$	set of polyhedra over set of variable set \mathbf{X} 8, 11, 45, 46
2^A	power-set of A 7, 9, 10, 12, 45, 46, 101
$\text{reach}(\mathcal{H})$	reachable set of hybrid automaton \mathcal{H} . . 14, 15, 26, 27, 44, 47, 51, 52, 86, 95, 98
$\text{rmax}_{\mathcal{H}}$	largest rate in rectangular hybrid automaton \mathcal{H} .33, 76–78, 80, 97
\models	satisfaction relation viii, xii, 14–16, 84, 86–90, 93, 94, 104
$\not\models$	negation of \models 87, 88
$\text{last}(\tau)$	last element of finite trajectory τ 13, 14, 76, 77, 86, 92, 97, 98
$\text{snd}(t)$	second element of tuple t 7, 13

$\text{succ}(e)$	successor of edge e 90
$A \rightarrow B$	the set of functions from A to B . 7, 9–12, 18, 20, 26, 27, 45, 46, 78, 91
B^A	same as $A \rightarrow B$ 7–9, 12, 56, 57, 59, 91

Chapter 1

Introduction

Cyber-Physical Systems are systems in which computer programs control physical environments through their sensors and actuators. They have applications in aerospace, self-driving cars, medical devices, smart grids, robotics, and more. Cyber-physical systems are becoming an indispensable part of our lives, and failures in these systems are often catastrophic financially and/or in terms of human life. Automakers recalled a record of 51.2 million vehicles over 868 separate recalls in 2015 for safety defects which cost them billions. Two million medical devices were recalled in the past decade out of which 24% were because of design bugs. In 2005, an overflow resulted in the flight computer crash, and in May 2016, a self-driving car that was equipped by sophisticated computer software, sensors, cameras, and radar, did not activate its brakes and caused the first fatal accident involving self-driving cars. Companies no longer have the luxury of time to spend decades on testing these systems manually. Therefore, it is more crucially important than ever to be able to *formally verify* these systems automatically and efficiently.

Whenever we want to *rigorously prove* that a cyber-physical system satisfies its required properties (*i.e.*, we want to formally verify), we first model it in some mathematical language. Hybrid automata [1] is a popular formal model for cyber-physical systems, in which continuous dynamics of the physical environment as well as discrete behavior of the controller are modeled and interact with each other. In addition to having a formal model of a cyber-physical system, one also needs to *formally* specify required properties. The most important property of a cyber-physical system is its *safety*, which as its name suggests, means we want our cyber-physical system to never be in an unsafe state (*e.g.*, room temperature should not become too high or too low, cars should not get too close to each other in the road, ...). A hybrid automaton models a state of a cyber-physical system as a tuple of discrete and continuous variables. A safety property first specifies an unsafe set of

states as a predicate on hybrid automata variables and then asks whether it is possible for the hybrid automaton in question to ever *reach* to this unsafe set. If the answer is *yes* then the system is *unsafe*, and if the answer is *no* then the system is *safe*. That is why safety and reachability are considered dual of each other. This reachability/safety problem has been carefully studied in the past couple of decades and boundaries of decidability have been extensively explored. The problem is stubbornly undecidable as shown by the many undecidability results in the area [2–6]. Results identifying decidable subclasses are few and rare. Apart from some low dimensional hybrid systems [4, 7–10], the main classes of decidable hybrid systems are timed automata [11], initialized rectangular automata [2], semi-algebraic o-minimal systems [12], and semi-algebraic STORMED systems [6].

There are couple of reasons for intrinsic computational complexity of model checking cyber physical systems. First, a cyber-physical system involves an interaction between a physical environment and computer software as controller. Even when a software has no interaction with a physical environment, its verification is in general undecidable. Consequently, model checking cyber-physical systems, even when the physical environment has a very simple dynamic, is undecidable. Second reason for the surprisingly high computational complexity of cyber-physical systems is the dynamics of entities in the physical environment. Often, ordinary differential equations (ODEs) are used to model these dynamics. However, decidability results are known only for very restrictive class of ODEs. The third reason is that in model checking cyber-physical systems, one usually assumes more than one initial state, and whenever this is the case, since entities in the physical environment mostly take continuous values, set of initial states will most likely be an uncountable set. Therefore, to verify a cyber-physical system, even if dynamics of the system was very simple and even if we were only interested in runs of bounded length/duration, one cannot test all the possible executions one by one. Note that this problem can never happen if we only consider bounded length possible executions of software in isolation. Finally, models of cyber-physical systems, most often, include non-deterministic behaviors. The non-determinism is added to models for two reasons. The first reason is that we may not know exactly what physical environment will do under certain condition. In this situation, we usually consider all the possibilities; the model specifies a set of possible behaviors and the cyber-physical system

may behave according to any of them. Therefore, if we can prove that the model is safe, since all the possible behaviors are considered, we conclude the cyber-physical system is safe as well. Note that even if we start from a single initial state, the non-determinism often introduces uncountable number of possible behaviors. Another reason for injecting non-determinism into the models, is to simplify the dynamics of deterministic models! Although, this simplification/non-determinism simplifies the dynamics and hence simplifies model checking the new models, the problem of having uncountable number of executions is there and, even in theory, prevents considering every possible behavior separately.

Given the computational difficulty of analyzing most hybrid systems, there are three approaches different researchers take to reduce the complexity of model checking. This thesis makes contributions in these three directions.

The first approach is considering *time bounded* variation of the reachability/safety problem (as opposed to the unbounded time). In this variation, in addition to a hybrid automaton and an unsafe set of states, there will be a time bound T as an input. Time-bounded reachability/safety problem asks whether it is possible to reach to an unsafe state *within time* T . Time-bounded variation has received much attention. It has been shown that time bounded problems in many cases are computationally easier than the corresponding problems without time bounds [13, 14]. For example, the time bounded reachability problem has been shown to be NEXPTIME-complete for monotonic rectangular automata, even though the same problem for non-monotonic rectangular hybrid automata is undecidable [15, 16].

The second approach to make model checking cyber-physical systems feasible, especially when there is no time bound, is to abstract continuous dynamics of the model. In this approach, continuous dynamics are abstracted into simpler dynamics that are amenable to automated analysis. This is because the general problem of safety verification is undecidable even for very simple class of continuous dynamics [2–6]. The success of the abstraction based method depends on finding the right abstraction, which can be difficult. One approach that tries to address this issue is the Counter Example Guided Abstraction Refinement (CEGAR) framework [17] that tries to automatically discover the right abstraction through a process of progressive refinement based on analyzing spurious counter examples in abstract models. CEGAR has been found to be useful in several contexts [18–21], including

hybrid systems [22–29]. Finally, whenever the goal is to *prove* safety, the abstraction (whether it is CEGAR-based or not) must *over approximate* all the possible behaviors in the system, and similarly whenever the goal is to *disprove* it, the abstraction must *under approximate* possible behaviors of them system. Therefore, if we manage to prove the abstract system is safe (unsafe) then we know the actual cyber-physical system is safe (unsafe) as well.

The third approach to make model checking cyber-physical systems feasible, looks deeper into undecidability results and other problems with the traditional modeling, verification, and implementation steps. Many undecidability proofs are carried out through the ability to efficiently encode/decode all natural numbers within the interval of real numbers from 0 to 1 (or any other pair of integers). This means there is no lower bound on the distance of real numbers with a corresponding natural number. Therefore, to correctly decode a real number into the corresponding natural number, its *exact* value, and not any approximation of it, is required. Proofs usually obtain this by assigning exactly one dynamic to each continuous variable within a mode of hybrid automaton, and use *equalities* in their hybrid automaton model specification. However, hybrid automata are supposed to model cyber-physical systems, and in a typical cyber-physical system, there is always some noise in the physical environment, which means we can never know the exact dynamics of the continuous variables. Furthermore, constraints in a hybrid automaton that do not specify continuous dynamics, usually model sensors in the cyber-physical system. Using equality in models means we assume sensors are infinitely precise, which is another false assumption. These two false assumptions (*i.e.* not having noise in the physical environment and having infinitely precise sensors) have at least one more negative consequence; they almost always make the implementation impossible. Suppose, we managed to prove that a hybrid automaton satisfies all its required properties. The next step in building a cyber-physical system is to implement the controller part of the hybrid automaton model. However, having exactly one continuous dynamics cannot be implemented, because time in a controller’s CPU is a discrete entity as opposed to time in the physical environment which is continuous. Furthermore, even if sensors are infinitely precise, the precision of data that is received by the controller is always finite. Note that these problems arise even for implementing timed automata, which perhaps con-

stitute the simplest class of hybrid automata. These difficulties may lead one to question the modeling language used for specifying different classes of hybrid automata and their problems. *Robust* model checking is solution to these problems that is suggested by researchers in [30–41]. In this approach, existence of noise in the environment and finite precision sensors are always assumed. However, the formal definitions of noise, environment, and precision, are different across different works.

1.1 Structure of the Dissertation

Chapter 2 is dedicated to preliminary concepts and definitions. In Chapter 3, we introduce a new class of hybrid automata called Initialized linear inclusion automata. In this class, invariants, guards, resets, and initial values are given by rectangular constraints, the flows are described by linear differential inclusions of the form $ax + b \triangleleft_1 \dot{x} \triangleleft_2 cx + d$ (with $\triangleleft_1, \triangleleft_2 : \{<, \leq\}$), and a variable x is reset on mode change whenever the differential inclusion describing the dynamics for x changes. Such automata strictly subsume initialized rectangular automata, which is known as a border of decidability results for hybrid automata [2]. For this class, we prove 1. unbounded time reachability is undecidable, 2. bounded time reachability is decidable. Note that bounding time does not necessarily bound number of discrete transitions that one can take within the given time.

Although, our algorithm for bounded time reachability in Chapter 3 is an interesting theoretical result, its application in practice is rather limited. Because, most cyber-physical systems in industry are not modeled using initialized linear inclusion automata. To deal with this problem, in Chapter 4, at the expense of losing decidability, we extend the class of hybrid automata and consider the problem of safety verification for non-linear polyhedral automata. In this class of hybrid automata, dynamics are given in the form of $\dot{x}_i := f_i(x_1, \dots, x_n)$ separately for every variable, where f_i is a non-linear function, and initial states, invariants, and reset relations are all given using polyhedral constraints. We present a Counter-Example Guided Abstraction Refinement Framework (CEGAR), which abstract these hybrid automata into polyhedral automata (*i.e.* initial states, invariants, flows, and reset relations are all given using polyhedral constraints). We show that the CEGAR

framework forming the theoretical basis of **HARE**, makes provable progress in each iteration of the abstraction-refinement loop. We analyze the performance of our tool against standard benchmark examples, and show that its performance is promising when compared to state-of-the-art safety verification tools, **SpaceEx** [42], **PHAVer** [43], **SpaceEx AGAR** [44], for affine dynamics and **HSolver** [45], **C2E2** [46], and **FLOW*** [47], for non-linear dynamics.

As we mentioned in the previous section, many undecidability results, including ours in Chapter 3, use some properties of modeling languages that have no correspondence in the real world. For example, our undecidability proof in Chapter 3, assumes there is absolutely no noise in the physical environment and sensors that obtain information from the physical environment have infinite precision. We also mentioned, many modeling languages designed for specification of cyber-physical systems, including the ones we use in Chapter 3 and Chapter 4, have the implementability issue. Knowing about these unrealistic properties of modeling languages and their negative consequences, Chapter 5 focuses on robust model checking. In this chapter, we consider the problem of robust model checking for monotonic rectangular automata (when time is bounded) and timed automata (when time is unbounded). These classes have an idealized semantics where clocks are assumed to be perfectly continuous and synchronized, and guards have infinite precision. These assumptions cannot be realized physically. To ensure that correct timed automata designs can be implemented on real-time platforms, several authors have suggested that timed automata be studied under robust semantics. A timed automaton \mathcal{H} is said to robustly satisfy a property if there is a positive ϵ and/or a positive δ such that the automaton satisfies the property even when the clocks can drift by ϵ and/or guards are enlarged by δ . In this chapter we show that, 1. checking ω -regular properties when only clocks are perturbed or when both clocks and guards are perturbed, is PSPACE-complete; and 2. one can compute the exact reachable set of a bounded timed automaton when clocks are drifted by infinitesimally small amount, using polynomial space. In particular, we remove the restrictive assumption on the timed automaton that its region graph only contains progress cycles, under which the second result above has been previously established. Finally, in Chapter 6, we discuss conclusions and future work.

Chapter 2

Preliminaries

This chapter introduces notations and mathematical definitions that have been used in the rest of the thesis. In Section 2.1 we introduce notions that are used for sets and functions. In Section 2.2 we introduce formal notions of transition systems, hybrid automata and its subclasses, and perturbation of hybrid automata.

2.1 Sets and Functions

The sets of *natural*, *positive natural*, *rational*, *positive rational*, *real*, *positive real*, and *non-negative real* numbers are represented by \mathbb{N} , \mathbb{N}_+ , \mathbb{Q} , \mathbb{Q}_+ , \mathbb{R} , \mathbb{R}_+ , and $\mathbb{R}_{\geq 0}$, respectively. For an element x and a set A , we use $x : A$ to denote that x is of type A ($x : A$ is *not* a predicate). Equalities written using symbol $:=$ are *definitions*. For $r : \mathbb{R}$, we define $\lfloor r \rfloor$ to be the largest integer not larger than r , and $\lceil r \rceil$ to be the smallest integer not smaller than r . When $r \geq 0$, we also define $\langle r \rangle$ to be $r - \lfloor r \rfloor$.

Let A and B be two arbitrary sets. Size of A is denoted by $|A|$. The power set of A is represented by 2^A . The Cartesian product of A and B is represented by $A \times B$. For a tuple $t := (a, b)$, elements a and b are represented by $\text{fst}(t)$ and $\text{snd}(t)$, respectively. If $A, B \subseteq \mathbb{R}$ then $A + B := \{a + b \mid a : A \wedge b : B\}$. The set of functions from A to B is represented by B^A or $A \rightarrow B$. For a function $f : A \rightarrow B$, $\text{dom}(f)$ denotes A , domain of f . For any two functions $f, g : A \rightarrow \mathbb{R}$, the function $f - g : A \rightarrow \mathbb{R} : x \mapsto f(x) - g(x)$ is the pointwise subtraction of g from f . For any $C \subseteq A$, $f[C \leftarrow 0] : A \rightarrow \mathbb{R}$ maps x to 0 if $x \in C$ and maps x to $f(x)$ if $x \notin C$. Consider $D \subseteq 2^B$. If D is closed under intersection and $f, g : A \rightarrow D$, $f \cap g : A \rightarrow D$ is the function $(f \cap g)(x) := f(x) \cap g(x)$. For functions $f : A \rightarrow D$ and $g : A \rightarrow B$, we say $g \dot{\in} f$ iff $\forall x : A \bullet g(x) \in f(x)$. For every function $x : [\mathbb{R}_{\geq 0} \rightarrow \mathbb{R}]$ that maps an input

t to $x(t)$, the first derivative with respect to t will be denoted by either $\frac{dx}{dt}$ or \dot{x} . If $\dot{x} = ax + b$ for some $a, b: \mathbb{R}$, then the solution is given by:

$$x(t) = \begin{cases} x(0)e^{at} + \frac{be^{at} - b}{a} & \text{if } a \neq 0 \\ x(0) + bt & \text{otherwise} \end{cases} \quad (2.1)$$

For any set A , and $n: \mathbb{N}$, $A^{\{0, \dots, n-1\}}$ is the set of *finite sequences on A of length n* . For any such sequence τ and index $i: \{0, \dots, n-1\}$, we may write τ_i instead of $\tau(i)$ to make the notation simpler. Knowing τ , we use $|\tau|$ to denote n . *Infinite sequences on A* is defined to be the set $A^{\mathbb{N}}$. If τ is an infinite sequence, $|\tau|$ is defined to be ∞ , and for any $i: \mathbb{N}$, we use τ_i to denote $\tau(i)$. For any set A , we use A^∞ to denote the set of finite sequences on A , and use A^ω to denote the set of infinite sequences on A .

For any $a: \mathbb{R} \cup \{-\infty\}$ and $b: \mathbb{R} \cup \{\infty\}$, $(a, b) := \{x: \mathbb{R} \mid a < x < b\}$ is the *interval* of real numbers between a and b . $(a, b]$, $[a, b)$, and $[a, b]$ are defined in a similar way (*i.e.* using bracket instead of parentheses means the corresponding end-point is closed). An interval is *closed* (*open*) if it can be written in the form of $[a, b]$ ((a, b)). It is *bounded* if both of its end-points belong to \mathbb{R} . We denote the set of *intervals* by \mathcal{I} . Also, the set of *bounded intervals* and *bounded closed intervals* are denoted by \mathcal{I}° and \mathcal{I}^\bullet , respectively. Note that \mathcal{I} , \mathcal{I}° , and \mathcal{I}^\bullet are closed under finite intersection, and intersection is easily computable (*i.e.* if end-points are given in binary formats and closeness/boundedness are specified using boolean flags, intersection of every two interval can be computed in constant amount of time).

A *polyhedron* (plural: polyhedra) is a finite intersection of constraints of the form $\sum_{x: X} c_x x \bowtie b$, where X is an arbitrary set of variables, $c_x: \mathbb{R}$ and $b: \mathbb{R}$ are arbitrary constants, and $\bowtie: \{<, \leq, =, \geq, >\}$ is the comparison type. We show the set of polyhedra defined on X by $\mathbb{P}(X)$. A *rectangle* is a polyhedron in which all constraints are of the form $x \bowtie b$.

2.2 Transition Systems and Hybrid Automata

In theoretical computer science, transition system is a concept usually used in the study of computation. It is basically a labeled directed graph in which number of nodes, labels, and edges could be even uncountable. It is quite

possible that using a single label one can go to multiple destinations from a single node. Therefore, transitions systems are often non-deterministic. In this thesis, we use transition systems to define the formal semantics of hybrid automata.

Definition 1 (Transition Systems). A transition system \mathcal{T} is a tuple $(S, \Sigma, \rightarrow, S^{\text{init}})$ in which

- S is a (possibly infinite) set of *states*,
- Σ is a (possibly infinite) set of *labels*,
- $\rightarrow \subseteq S \times \Sigma \times S$ is a *transition relation*, and
- $S^{\text{init}} \subseteq S$ is a set of *initial states*.

We denote different elements of \mathcal{T} by adding a subscript to their names. For example, we use $S_{\mathcal{T}}$ to denote the set of states of \mathcal{T} . We may omit the subscript whenever it is clear from the context. We write $s \xrightarrow{\sigma} s'$ instead of $(s, \sigma, s') \in \rightarrow$, and $s \rightarrow s'$ to denote $\exists \sigma : \Sigma . s \xrightarrow{\sigma} s'$.

A hybrid automaton (plural: hybrid automata) is a mathematical model we use throughout this thesis to formally model a cyber-physical system. It has a finite number of locations and a finite number real variables. In each location, values of variables continuously change according to some given dynamics. Starting from some initial location, one can go to different locations if there is an edge between source and destination locations. Furthermore, in order to take an edge, values of variables must satisfy a transition relation associated to each edge. Definition 2 formally specifies syntax of a hybrid automaton.

Definition 2 (Hybrid Automata). A hybrid automaton \mathcal{H} is a tuple $(Q, X, I, F, E, X^{\text{init}})$ in which

- Q is a non-empty finite set of *locations*.
- X is a non-empty finite set of *variables*. We let $V := \mathbb{R}^X$ be the set of all possible *valuations* of variables in X .
- $I : Q \rightarrow 2^V$ maps each location to the set of possible valuations in that location as *invariant* of the location.

- $F: \mathbb{Q} \rightarrow 2^{\mathbb{V} \times \mathbb{V}}$ maps each location q to the set of possible *flows* of that location. Each element in this set is a pair $(\nu, \dot{\nu})$. Intuitively it means, if the current continuous state is ν then $\dot{\nu}$ is a possible direction field.
- E is a set of *edges* of the form $e = (s, d, l, r)$ where
 - $s: \mathbb{Q}$ is the *source* of e ,
 - $d: \mathbb{Q}$ is the *destination* of e ,
 - $l: 2^E$ is the *label* of e ¹, and
 - $r: 2^{\mathbb{V} \times \mathbb{V}}$ is the *reset* relation of e .

We let $G(e) := \{\nu: \mathbb{V} \mid \exists \nu': \mathbb{V} \bullet (\nu, \nu') \in r\}$ to be *guard* of e , as the set of valuations for which the reset relation is non-empty. We use $S(e)$, $D(e)$, $L(e)$, $R(e)$, and $G(e)$, to denote different elements of edge e .

- $X^{\text{init}}: \mathbb{Q} \rightarrow 2^{\mathbb{V}}$ maps each location to the set of possible *initial valuations* in that location. We let $Q^{\text{init}} := \{q: \mathbb{Q} \mid \exists \nu: \mathbb{V} \bullet (q, \nu) \in X^{\text{init}}\}$ to be the set of *initial locations*, *i.e.* those locations for which there is at least one initial valuation.

We denote different elements of \mathcal{H} by adding a subscript to their names. For example, we use $X_{\mathcal{H}}$ to denote the set of variables of \mathcal{H} . We may omit the subscript whenever it is clear from the context.

Definition 3 (Semantics of Hybrid Automata). The semantics of a hybrid automaton \mathcal{H} is defined using the transition system $\llbracket \mathcal{H} \rrbracket = (S, \Sigma, \rightarrow, S^{\text{init}})$ in which

- $S := \mathbb{Q} \times \mathbb{V}$,
- $\Sigma := E \cup \mathbb{R}_{\geq 0}$ ²,
- $S^{\text{init}} := \{(q, \nu) \mid \nu \in X^{\text{init}}(q) \cap I(q)\}$, and
- $\rightarrow := \rightarrow_1 \cup \rightarrow_2$ where

¹in this thesis we do *not* compose different hybrid automata. Edges having labels as subset of edges are used in some of the proofs when we want to merge edges while remembering original edges that are merged. Also, implicit recursive definition is not a problem, instead of l being a subset of E , one can order E and let l be a subset of $\{0, \dots, |E| - 1\}$.

²*Wlog.* we assume E and \mathbb{R} are disjoint.

- \rightarrow_1 is the set of *continuous* transitions and for all $t : \mathbb{R}_{\geq 0}$ we have $(q, \nu) \xrightarrow{t}_1 (q', \nu')$ iff $q = q'$ and there is a function $f : [0, t] \rightarrow \mathbb{V}$ with a free variable u such that 1. $f(0) = \nu$, 2. $f(t) = \nu'$, 3. $\forall u : [0, t] \bullet f(u) \in \mathbf{I}(q)$, and 4. $\forall u : [0, t] \bullet \left(f(u), \frac{df}{dt}(u)\right) \in \mathbf{F}(q)$. When the free variable of the function f is known, we use \dot{f} to refer to the first derivative of f with respect to its free variable ³.
- \rightarrow_2 is the set of *discrete* transitions and for any $e : \mathbf{E}$ we have $(q, \nu) \xrightarrow{e}_2 (q', \nu')$ iff 1. $q = \mathbf{S}(e)$, 2. $q' = \mathbf{D}(e)$, 3. $\nu \in \mathbf{I}(q)$, 4. $\nu' \in \mathbf{I}(q')$, and 5. $(\nu, \nu') \in \mathbf{R}(e)$.

Every state in the semantics is a pair of discrete and continuous states. If we spend time in a location, it means the discrete component of the transition won't change. The continuous part, however, will change according to some possible dynamics for the current location. There is no bound on how long one can stay in a location, but as long as location is fixed, its invariant must be satisfied. Taking a discrete edge could change both components of the current state. Discrete part changes from source to destination location, and continuous part will non-deterministically change to any value that satisfies the transition relation.

2.2.1 Subclasses of Hybrid Automata

We consider different sub-classes of hybrid automata.

- *Polyhedral automata* are hybrid automata, in which for any location $q : \mathbf{Q}$ values of $\mathbf{I}(q)$, $\mathbf{F}(q)$, and $\mathbf{X}^{\text{init}}(q)$ are specified using polyhedra in $\mathbb{P}(\mathbf{X})$. Also, for any edge $e : \mathbf{E}$, $\mathbf{R}(e)$ is specified using polyhedra in $\mathbb{P}(\mathbf{X} \cup \mathbf{X}')$, where \mathbf{X}' is the prime version of \mathbf{X} (*i.e.* for any $x_1, x_2 : \mathbf{X}$ we assume $x_1 \neq x_2 \Leftrightarrow x'_1 \neq x'_2$, $x'_1 \in \mathbf{X}'$, and $|\mathbf{X}| = |\mathbf{X}'|$). A polyhedron $P : \mathbb{P}(\mathbf{X})$ defines $\{(\nu, \nu') \mid \nu' \in P\}$ as the flow. Therefore, $\mathbf{F}(q)$ is independent of the current valuation. We abuse the notation and write $\mathbf{F}(q) = P$ when it causes no confusion. The same is true for $\mathbf{I}(q)$, $\mathbf{X}^{\text{init}}(q)$, and $\mathbf{R}(e)$. Finally, we require that only non-empty polyhedra be used in the specification of hybrid automata.

³We only consider left/right derivatives of f at the right/left end points of $[0, t]$.

- *Non-linear polyhedral automata* are same as polyhedral automata except that for any location $q : \mathbb{Q}$ value of $F(q)$ is specified by a function f of type $I(q) \rightarrow \mathbb{V}$. Therefore, for any valuation in the invariant of that location, $F(q)$ defines exactly one execution. We abuse the notation and write $F(q) = f$ when it causes no confusion. In this thesis, we always assume function f is Lipschitz continuous.
- *Rectangular automata* are polyhedral automata in which for any location $q : \mathbb{Q}$ values of $I(q)$, $F(q)$, and $X^{\text{init}}(q)$ are specified using by rectangles in $\mathbb{I}^{\mathbb{X}}$. Also, for any edge $e : \mathbb{E}$ value of $R(e)$ is specified using three elements: 1. $g : \mathbb{I}^{\mathbb{X}}$ as the guard of e , 2. $j : 2^{\mathbb{X}}$ as the jump set of e , and 3. $r : \mathbb{I}^j$ as the reset of e . Polyhedron $R(e)$ is defined using the union of three constraints: 1. $\bigwedge_{x : \mathbb{X}} x \in g(x)$, and 2. $\bigwedge_{x : j} x' \in r(x)$, and 3. $\bigwedge_{x : \mathbb{X} \setminus j} x' = x$. Intuitively, constraint g must be satisfied in order to take edge e . After taking e , values of variables in j are non-deterministically reset to some value in $r(x)$ and values of other variables don't change. We denote j and r by respectively $J(e)$ and $R^{\square}(e)$. Note that $G(e)$ in Definition 2, is exactly g . A rectangular automaton is called *initialized* iff for any edge $e : \mathbb{E}$ and $x : \mathbb{X}$ if $F(S(e))(x) \neq F(D(e))(x)$ then $x \in J(e)$. A rectangular automaton is called *closed* iff all intervals in its specification are closed. A rectangular automaton is called *monotonic* iff for any variable x , possible flows of x is either non-negative in all locations or non-positive in all locations. Finally, in order to make the notation simpler, we may write $I(q, x)$ instead of $I(q)(x)$. The same is true for $F(q, x)$, $X^{\text{init}}(q, x)$, $G(e, x)$, and $R^{\square}(e, x)$.
- *Stopwatch automata* are restricted class of rectangular automata where $F(q, x)$ could be either $[0, 0]$ and $[1, 1]$. Furthermore, $X^{\text{init}}(q, x)$ and $R^{\square}(e, x)$ must be intervals of the form $[r, r]$, for some $r : \mathbb{R}$.
- *Linear inclusion automata* are same as rectangular automata except for any location $q : \mathbb{Q}$ value of $F(q)$ is specified using $\bigwedge_{x : \mathbb{X}} a_x x + b_x \triangleleft_{x,l} x \triangleleft_{x,u} c_x x + d_x$, where $a_x, c_x : \mathbb{R}$, $b_x : \mathbb{R} \cup \{-\infty\}$, $d_x : \mathbb{R} \cup \{\infty\}$, and $\triangleleft_{x,l}, \triangleleft_{x,u} : \{<, \leq\}$. The set $\{(\nu, \nu') \mid \forall x : \mathbb{X} \bullet a_x \nu_x + b_x \triangleleft_{x,l} \nu'_x \triangleleft_{x,u} c_x \nu_x + d_x\}$ defines flow of q . Note that linear inclusion automata are not a sub-class polyhedral automata because flows are dependent on current valuations. Nor are they a sub-class of non-linear polyhedral automata because flows are

given in the form of inclusions not equations. Initialized linear inclusion automata are defined in the same manner as rectangular automata (*i.e.* if source and destination locations of edge e have different flow for a variable x than x must be in $J(e)$).

- *Timed automata* are a sub-class of initialized rectangular automata, where for any location $q : \mathbf{Q}$, edge $e : \mathbf{E}$, and variable $x : J(e)$, the following three conditions hold: 1. $F(q, x) := [1, 1]$, 2. $R^\square(e, x) := [0, 0]$, and 3. $X^{\text{init}}(q, x)$ is either \emptyset or $[0, 0]$. We call variables of a timed automaton *clocks*. A timed automaton is called *closed* iff all intervals in its specification (*i.e.* guards and initial values) are closed.

2.2.2 Trajectories and Executions

In this section, we define trajectories, executions, and different operations on them. We use these to finally define reachable sets as well as the two main problems that are the focus of this thesis: 1. safety verification problem, and 2. ω -regular model checking problem.

For any hybrid automaton \mathcal{H} with the semantics $\llbracket \mathcal{H} \rrbracket = (\mathbf{S}, \Sigma, \rightarrow, \mathbf{S}^{\text{init}})$, an element τ from the set $\mathbf{S} \times (\Sigma \times \mathbf{S})^\omega$, and an index $n : \mathbb{N}$, we let τ_n denote $\text{fst}(\tau)$ if $n = 0$ and $\text{snd}(\text{snd}(\tau)_{n-1})$ if $n > 0$. Also $L_n(\tau)$ will denote $\text{fst}(\text{snd}(\tau)_n)$. Furthermore, if $\text{snd}(\tau) \in (\Sigma \times \mathbf{S})^\omega$, we define $|\tau|$ to be ∞ . Otherwise, we define $|\tau|$ to be $|\text{snd}(\tau)|$ (*i.e.* length of the sequence $\text{snd}(\tau)$). Note that τ_n is defined only if $n \leq |\tau|$ and $L_n(\tau)$ is defined only if $n < |\tau|$. Finally, we define $\text{dom}(\tau)$ to be $\{0, 1, \dots, |\tau| - 1\}$. Note that τ is completely known if we specify $|\tau|$, τ_n , and $L_n(\tau)$, for all defined indices (recall that defined indices for τ_n are $\{0, 1, \dots, |\tau|\}$ and for $L_n(\tau)$ are $\{0, 1, \dots, |\tau| - 1\}$).

τ is said to be a *trajectory* iff for any $n : \text{dom}(\tau)$ we have 1. $\tau_n \xrightarrow{L_n(\tau)} \tau_{n+1}$, and 2. if $n > 0$ then $L_n(\tau) \in \mathbb{R}_{\geq 0}$ iff $L_{n-1}(\tau) \notin \mathbb{R}_{\geq 0}$ (*i.e.* τ has alternating sequence of discrete and continuous transitions). The set of *trajectories*, *finite trajectories*, and *infinite trajectories* of \mathcal{H} is denoted by $\llbracket \mathcal{H} \rrbracket_\infty$, $\llbracket \mathcal{H} \rrbracket_*$, and $\llbracket \mathcal{H} \rrbracket_\omega$, respectively. We use $\text{first}(\tau)$ to refer to τ_0 . Also, if $|\tau|$ is finite, $\text{last}(\tau)$ refers to $\tau_{|\tau|}$. For any trajectory $u : \llbracket \mathcal{H} \rrbracket_*$ and $v : \llbracket \mathcal{H} \rrbracket_\infty$, such that 1. $\text{last}(u) = \text{first}(v)$, and 2. $|u|, |v| > 0$ or $L_{|u|-1}(u) \in \mathbb{R}_{\geq 0} \Leftrightarrow L_0(v) \notin \mathbb{R}_{\geq 0}$ (*i.e.* u ends in a time transition iff v starts with discrete one), the *concatenation* of u and v is a trajectory $\tau : \llbracket \mathcal{H} \rrbracket_\infty$, denoted by $u \sim v$, specified as follows: 1. $|\tau| = |u| + |v|$ (τ

is an infinite trajectory iff v is), 2. $\tau_n := u_n$ if $n \leq |u|$ and $\tau_n := v_{n-|u|}$ otherwise, and 3. $L_n(\tau) := L_n(u)$ if $n < |u|$ and $L_n(\tau) := L_{n-|u|}(v)$ otherwise. A trajectory τ is said to be an *execution* iff $\text{first}(\tau) \in \mathbf{S}^{\text{init}}$, i.e., it starts from an initial state. We denote the set of *executions*, *finite executions*, and *infinite executions* of \mathcal{H} by $\llbracket \mathcal{H} \rrbracket_\infty^0$, $\llbracket \mathcal{H} \rrbracket_*^0$, and $\llbracket \mathcal{H} \rrbracket_\omega^0$, respectively. We define $\text{reach}(\mathcal{H})$ to be $\{\text{last}(\tau) \mid \tau : \llbracket \mathcal{H} \rrbracket_*^0\}$, it simply is the set of states in $\llbracket \mathcal{H} \rrbracket$ that one can reach from an initial state of $\llbracket \mathcal{H} \rrbracket$ in finite amount of time. Similarly, for any time bound $T : \mathbb{R}_{\geq 0}$, we define $\text{reach}(\mathcal{H}, T)$ to be $\{\text{last}(\tau) \mid \tau : \llbracket \mathcal{H} \rrbracket_*^0 \wedge \text{duration}(\tau) \leq T\}$, it simply is the set of states in $\llbracket \mathcal{H} \rrbracket$ that one can reach from an initial state of $\llbracket \mathcal{H} \rrbracket$ within time T . Finally, for any trajectory τ , we let $\text{inf}(\tau) \subseteq \mathbf{E}_{\mathcal{H}}$ be the set of edges that are visited infinitely often by τ .

Problem 4 (Safety/Reachability). For a hybrid automaton \mathcal{H} , let $\varphi \subseteq \mathbf{S}_{\mathcal{H}}$, be a set of unsafe states in \mathcal{H} . The *(unbounded-time) safety problem* asks whether or not the unsafe set φ has a non-empty intersection with $\text{reach}(\mathcal{H})$. In this thesis, we answer this question by computing $\text{reach}(\mathcal{H})$ and checking the emptiness of the intersection, hence we may refer to the same problem as the *reachability problem*. We also consider the bounded-time variation of this problem. The *bounded-time safety problem* asks whether or not the unsafe set φ has a non-empty intersection with $\text{reach}(\mathcal{H}, T)$, where $T : \mathbb{R}_{\geq 0}$ is another input to the problem.

Problem 5 (ω -Regular Model Checking). For a hybrid automaton \mathcal{H} , let $\varphi \subseteq \mathbf{E}_{\mathcal{H}}$, be a subset of edges in \mathcal{H} . The *ω -regular model checking problem* asks whether or not all infinite executions of \mathcal{H} visit some edge in φ infinitely often. More precisely, it asks if $\forall \tau : \llbracket \mathcal{H} \rrbracket_\omega^0 \bullet \text{inf}(\tau) \cap \varphi \neq \emptyset$ is true.

Taking φ to be both subset of states and edges of \mathcal{H} will not cause any confusion. In case φ is a subset of states, we write $\mathcal{H} \models \varphi$ to denote that the system is safe. And, in case φ is a subset of edges, we write $\mathcal{H} \models \varphi$ to denote that any execution of \mathcal{H} visits some edge in φ infinitely often.

2.2.3 Perturbation of Hybrid Automata

In order to *robustly* model check a hybrid automaton we need to define two types of perturbation. Perturbation and robust model checking are only used in Chapter 5 and that chapter only deals with rectangular automata with

closed intervals and its subclasses. Therefore, in this section we only consider rectangular automata with closed intervals.

For any $\epsilon, \delta : \mathbb{R}_{\geq 0}$ and closed rectangular automaton \mathcal{H} , we consider two types of perturbations. We use $\mathcal{H}_\delta^\epsilon$ to denote another rectangular automaton where guards and flows are enlarged by δ and ϵ , respectively. On the other hand, $\mathcal{H}_{+\delta}$ refers to another rectangular automaton where flows are not changed, but *positive* guards are perturbed by δ . Before formally defining these perturbations, we introduce some notation. For an interval $J = [a, b]$ and $\delta : \mathbb{R}_{\geq 0}$, $J_\delta := [a - \delta, b + \delta]$. On the other hand, $J_{+\delta} := [c, d]$, where

$$c = \begin{cases} a - \delta & \text{if } a > 0 \\ a & \text{otherwise} \end{cases} \quad d = \begin{cases} b + \delta & \text{if } b > 0 \\ b & \text{otherwise} \end{cases}$$

$\mathcal{H}_\delta^\epsilon$ is rectangular automaton $(\mathbf{Q}, \mathbf{X}, \mathbf{I}', \mathbf{F}', \mathbf{E}', \mathbf{X}^{\text{init}})$ where for any $q : \mathbf{Q}$, $x : \mathbf{X}$, and $(s, d, l, g, j, r) : \mathbf{E}$, the following are all true: 1. $(s, d, l, g_\delta, j, r) \in \mathbf{E}'$, 2. $\mathbf{I}'(q, x) := (\mathbf{I}(q, x))_\delta$, and 3. $\mathbf{F}'(q) := (\mathbf{F}(q))_\epsilon$. Rectangular automaton $\mathcal{H}_{+\delta}$ is same as $\mathcal{H}_\delta^\epsilon$ except that in $\mathcal{H}_{+\delta}$ we enlarge intervals using $J_{+\delta}$ instead of J_δ . We write \mathcal{H}^ϵ , \mathcal{H}_δ , and \mathcal{H} instead of \mathcal{H}_0^ϵ , \mathcal{H}_δ^0 , and \mathcal{H}_0^0 . Finally, we define the following sets of states representing reachable sets after different yet infinitesimal amount of perturbations.

$$\text{limreach}^\epsilon(\mathcal{H}) := \bigcap_{\epsilon : \mathbb{R}_+} \text{reach}(\mathcal{H}^\epsilon) \quad (2.2)$$

$$\text{limreach}_\delta(\mathcal{H}) := \bigcap_{\delta : \mathbb{R}_+} \text{reach}(\mathcal{H}_\delta) \quad (2.3)$$

$$\text{limreach}_\delta^\epsilon(\mathcal{H}) := \bigcap_{\epsilon : \mathbb{R}_+} \bigcap_{\delta : \mathbb{R}_+} \text{reach}(\mathcal{H}_\delta^\epsilon) \quad (2.4)$$

Problem 6 (Robust Safety/Reachability Problem). For a hybrid automaton \mathcal{H} , let $\varphi \subseteq \mathbf{S}_\mathcal{H}$, be a set of unsafe states in \mathcal{H} . There are four versions of *robust safety problem* defined as follow:

1. $\exists \epsilon : \mathbb{R}_+ \bullet \mathcal{H}^\epsilon \models \varphi$, robust safety when only flows are perturbed,
2. $\exists \delta : \mathbb{R}_+ \bullet \mathcal{H}_\delta \models \varphi$, robust safety when only guards are enlarged,
3. $\exists \delta : \mathbb{R}_+ \bullet \mathcal{H}_{+\delta} \models \varphi$, robust safety when only positive guards are enlarged,
and

4. $\exists \epsilon, \delta : \mathbb{R}_+ \bullet \mathcal{H}_\delta^\epsilon \models \varphi$, robust safety when both guards and clocks are perturbed.

Intuitively, these conditions look for robustness under different kinds of perturbation. Similar to non-robust version, we may refer to this problem as the *reachability problem* as well. Time-bounded variation is defined similarly.

Problem 7 (Robust ω -Regular Model Checking). For a hybrid automaton \mathcal{H} , the *robust ω -regular model checking problem* asks to identify which of the properties in Problem 6 are true when φ is a subset of edges.

Chapter 3

Exact Model Checking

The reachability problem for hybrid automata [1] is very important from the standpoint of safety verification of cyber-physical systems. This problem has been carefully studied in the past couple of decades and boundaries of decidability have been extensively explored. The problem is stubbornly undecidable as shown by the many undecidability results in the area [2–6]. Results identifying decidable subclasses are few and rare. Apart from some low dimensional hybrid systems [4, 7–10], the main classes of decidable hybrid systems are timed automata [11], initialized rectangular automata [2], semi-algebraic o-minimal systems [12], and semi-algebraic STORMED systems [6].

Given the computational difficulty of analyzing most hybrid systems, time bounded versions of classical decision problems have received much attention. It has been shown that time bounded problems in many cases are computationally easier than the corresponding problems without time bounds [13, 14]. One particular problem that has been investigated recently is the time bounded reachability problem, which asks if a certain state of a hybrid automaton can be reached within a given time bound T . The time bounded reachability problem has been shown to be NEXPTIME-complete for monotonic rectangular automata, even though the same problem for non-monotonic rectangular automata is undecidable [15, 16].

In this chapter, we consider a rather new class of hybrid automata called initialized linear inclusion automata. Like initialized rectangular automata, the invariants, guards, resets, and initial values in such automata are described by rectangular constraints, and variables are *initialized*, *i.e.*, whenever the continuous dynamics of a variable changes due to a mode switch, its value is required to be reset to a value in an interval range. However, unlike rectangular automata, the continuous dynamics is given by *linear differential inclusions* of the form $ax + b \triangleleft_1 \dot{x} \triangleleft_2 cx + d$ (where $\triangleleft_1, \triangleleft_2 : \{<, \leq\}$)¹. In other

¹Linear inclusions for each variable are scalar.

words, the evolution of a continuous variable x is any trajectory $x : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}$ such that at any time t , $ax(t) + b \triangleleft_1 \dot{x}(t) \triangleleft_2 cx(t) + d$. Thus, such automata strictly subsume the class of initialized rectangular automata. We anticipate such automata to be useful in abstracting hybrid automata more precisely than initialized rectangular automata. Evidence of such an application can be seen in the use of eigenforms for abstracting linear systems [48].

We show that the reachability problem for initialized linear inclusion automata is undecidable by reducing the halting problem of 2-counter machines. In addition, the time bounded reachability for (uninitialized) linear inclusion automata is undecidable. This follows from the undecidability of the time bounded reachability problem for (non-monotonic) rectangular automata [15, 16]. In contrast, we show that the time bounded reachability problem is decidable. Our decidability result is proved based on the following observations. Like to the translation of initialized rectangular automata timed automata [2, 49], we first reduce the reachability problem of initialized linear inclusion automata to the reachability of problem in an automaton all of whose continuous variables are clocks. Thus, we generalize an observation about rectangular flows to linear inclusion flows. The resulting automaton though is not a timed automaton because the constants used in the constraints could be of the form $r \ln r'$, where r, r' are rational numbers. We call such automata irrational timed automata. This difference is significant because reachability for such automata is undecidable; this is a consequence of our undecidability result for initialized linear inclusion automata. However, we show that the time bounded reachability for such automata is decidable. Note, that our decidability result does not follow from the result in [16] — while clocks are special monotonic rectangular variables, the presence of irrational constants in our automata complicates matters. Our decidability proof relies on observing that if a state ν is reachable within time T , it is reachable by an execution with at most exponentially many discrete transitions. The algorithm deciding time bounded reachability then guesses such an execution, and checks if the execution is valid. To check validity of an execution, we reduce the problem of checking negative cost cycles in an exponentially sized graph. The presence of irrational constants in irrational timed automata ensures that checking for negative cost cycles involves comparing linear combinations of natural logarithms of rational numbers with integers. All steps of our algorithm, except the step of comparing logs with integers,

can be bounded by PSPACE. Even though natural logs can be approximated very efficiently (both in terms of space and running time) with arbitrary precision [50–52], we are unaware of any complexity bounds for computing a particular bit (say k) of the natural logarithm of a rational number. This prevents us from proving hard upper bounds. We conjecture that the problem is in fact PSPACE-complete.

3.1 Unbounded Time Reachability

In this section, we consider the problem of (unbounded time) reachability in initialized linear inclusion automata and prove that it is in fact undecidable. Our proof is based on reducing the halting problem of 2-counter machine to the reachability problem for initialized linear inclusion automata. It is an adaptation of Miller’s proof [53] showing that the reachability problem for timed automata with irrational constants is undecidable. The reason why the construction needs to be changed is because Miller’s automata compare variables with both rational and irrational constants without resetting them between the two comparisons. However, the initialization requirement of initialized linear inclusion automaton forces one to reset variables when it is compared to different constants. We begin by defining the model of 2-counter machines.

Definition 8 (2-Counter Machine). A 2-counter machine is a program with counters that can be either incremented or decremented; the control flow in the program can be changed by testing if one of the counters is zero. Formally, we assume a countable set \mathcal{L} of labels and give a program with counters c_1, c_2 by the following BNF grammar:

$$\begin{aligned}
S ::= & \ell : \text{inc}(c) && (\text{increment counter } c) \\
& | \ell : \text{dec}(c) && (\text{decrement counter } c \text{ if } c > 0) \\
& | \ell : \text{jz}(c, \ell') && (\text{jump to statement } \ell' \text{ if } c = 0) \\
& | \ell : \text{halt} && (\text{halt}) \\
& | S; S && (\text{sequence})
\end{aligned}$$

where $\ell, \ell' : \mathcal{L}$, all ℓ s must be unique, and c is either c_1 or c_2 .

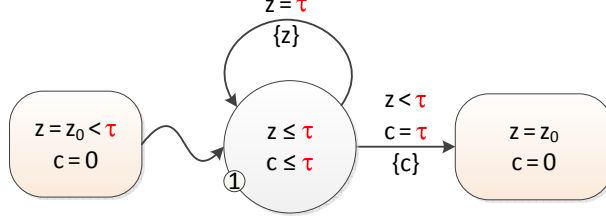
The formal semantics of 2-counter machines is skipped since it is standard.

To simplify the proof, we assume that all programs we consider have the property that no statement of the form $\text{dec}(c)$ (for $c: \{c_1, c_2\}$) can be reached in a context where $c = 0$. Note that it is easy to transform any program to one that meets this restriction, by including a zero test before every decrement; this causes the size of the program to only blow-up by a constant factor.

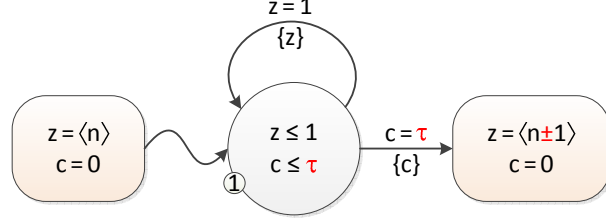
Before describing how a 2-counter machine will be simulated by an initialized linear inclusion automaton, we define an encoding of natural numbers that we will use in our reduction. Let $\langle \cdot \rangle : \mathbb{N} \rightarrow [0, 1) : n \mapsto \frac{r}{2^{\lfloor \lg r \rfloor}} - 1$, where $r := 3^n$ and \lg is logarithm with base 2. Thus, $\langle n \rangle$ maps n to the number $\frac{3^n}{2^i} - 1$, where i is the largest natural number such that $2^i \leq 3^n$. Observe that for any $n, m : \mathbb{N}$, $\langle n \rangle = \langle m \rangle$ iff $n = m$, and $\langle n \rangle = 0$ iff $n = 0$.

Given a 2-counter machine \mathcal{M} (with counters c_1 and c_2) our reduction will construct an initialized linear inclusion automaton \mathcal{A} with 3 variables x, y , and c . The variables x, y will be used to keep track of the values of the counters c_1 and c_2 , respectively, while c is an auxiliary variable used by \mathcal{A} . The intuition behind the construction is as follows. Each step of \mathcal{M} will be simulated in $w := \ln 16$ time units. Moreover, if after i^{th} step the values of counters c_1 and c_2 are m and n , respectively, then the valuation ν of \mathcal{A} at time $i \times w$ will be $\nu(x) = \langle m \rangle$, $\nu(y) = \langle n \rangle$, and $\nu(c) = 0$; the label of the statement to be executed next, will be stored in the location of \mathcal{A} . For any variable $v : \{x, y, c\}$, the dynamics in any control location is given by $\dot{v} = v + 1$. Thus, the automaton \mathcal{A} is trivially initialized, as all the variables have the same dynamics in all control states.

The crux of the construction can be understood by two gadgets shown in Figure 3.1. Recall that the value of a variable z , evolving with dynamics $\dot{z} = z + 1$, at time t is given by $(z_0 + 1)e^t - 1$, where z_0 is the value of z at time 0. In the gadgets shown in Figure 3.1, z denotes either variable x or y of \mathcal{A} . The gadget in Figure 3.1a ensures that if you enter the control state with z having value z_0 , and c being 0 then you leave the control state at time $\ln(\tau + 1)$ with z and c having the same value as at the beginning, for any $\tau > z_0$. This can be understood as follows. Since c is initially 0, and we exit the location when $c = \tau$, the total time spent is $\ln(\tau + 1)$. Now, the variable z is reset at time t when $z(t) = (z_0 + 1)e^t - 1 = \tau$, which means z is reset at time $t = \ln \frac{\tau+1}{z_0+1}$. In the remaining time $(\ln(\tau + 1) - \ln \frac{\tau+1}{z_0+1} = \ln(z_0 + 1))$, the value of z will return back to z_0 . The gadget in Figure 3.1b ensures that if $z = \langle n \rangle$ and $c = 0$ initially, then on leaving $z = \langle n + 1 \rangle$ if $\tau = 2$ or



(a) Waiting for $\ln(\tau + 1)$ units of time without changing the value of c or z , where $z_0 < \tau$.



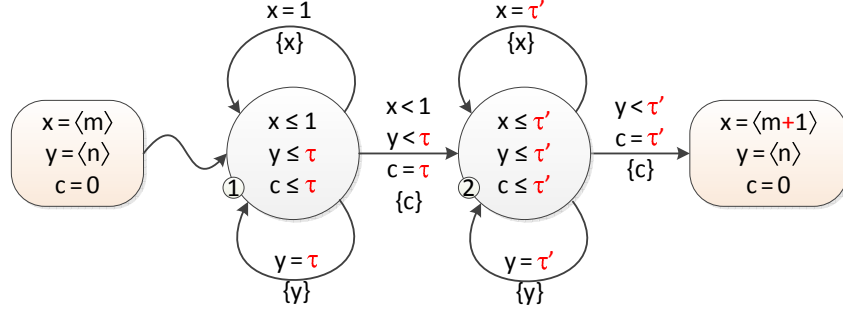
(b) inc (and dec): Incrementing the counter z when $\tau = 2$ and decrementing it when $\tau = \frac{13}{3}$.

Figure 3.1: Basic gadgets for simulating 2-counter machines. Dynamics of all variables $v : \{z, c\}$ is defined to be $\dot{v} = v + 1$.

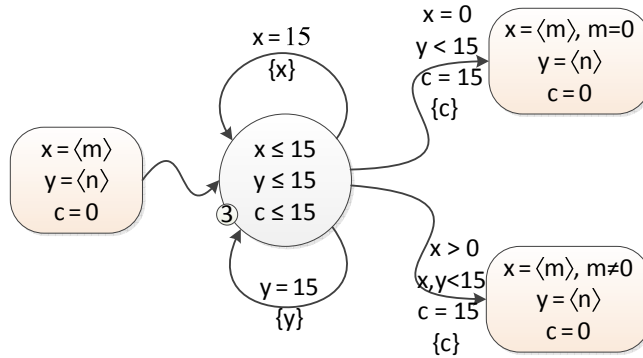
$z = \langle n - 1 \rangle$ (with $n > 0$) if $\tau = \frac{13}{3}$. We describe the intuition behind the increment; the decrement case is similar and skipped. We have $\tau = 2$. Since c is initially 0, and is 2 at exit, the total time spent in this control location is $\ln 3$ time units. Let us assume that z is initially $\frac{3^n}{2^i} - 1$. After $\ln \frac{2^{i+1}}{3^n}$ units of time, the value of z will be 1, and so the self loop transition will be taken and z will be reset to 0. It takes another $\ln 2$ units of time before z reaches 1 again. If $\ln 3 - \ln \frac{2^{i+1}}{3^n} = \ln \frac{3^{n+1}}{2^{i+1}}$ is greater than $\ln 2$ then z will be reset one more time. Assuming z is reset k times (where $k = 1$ or 2), the value of z at the time of leaving will be $\frac{3^{n+1}}{2^{i+k}} - 1$, which is $\langle n + 1 \rangle$.

We can combine and modify the above gadgets in different ways to obtain the gadgets that simulate each statement of a 2-counter machine. These simulation gadgets are shown in Figure 3.2. Thus, the 2-counter machine \mathcal{M} reaches the **halt** statement iff the initialized linear inclusion automaton \mathcal{A} constructed based on the above intuition reaches the location corresponding to the **halt** statement of \mathcal{M} . This proves that the control state reachability problem is undecidable for initialized linear inclusion automata.

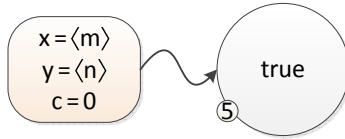
Theorem 9. Unbounded time reachability for initialized linear inclusion automata is undecidable.



(a) inc (and dec): Incrementing the counter x (final value of x would be $\langle m + 1 \rangle$). If we switch τ and τ' the result gadget decrements m (final value of x would be $\langle m - 1 \rangle$).



(b) jz: Comparing the counter x with 0. We know that $m = 0$ iff $x = 0$.



(c) halt: A halt location acts as a sink. No discrete transitions are enabled from the sink location.

Figure 3.2: Gadgets that simulate 2-counter machines. Parameters τ and τ' are 2 and $\frac{13}{3}$, respectively. Counters are x and y that are simulated respectively by variables x and y . Dynamics of all variables $z : \{x, y, c\}$ is defined to be $\dot{z} = z + 1$. Each state in the machine is replaced by one of these gadgets, based on the command in that state.

3.2 Translating Initialized Linear Inclusion Automata to Irrational Timed Automata

Although the reachability problem is undecidable for initialized linear inclusion automata (Theorem 9), in this section we show that time bounded reachability is decidable. We begin by observing that the reachability prob-

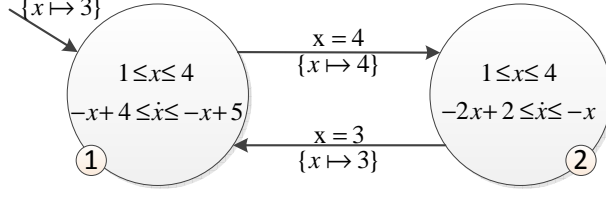


Figure 3.3: Example Initialized Linear Inclusion Automaton \mathcal{A}

lem (bounded and unbounded) for initialized linear inclusion automata can be reduced to the reachability problem (bounded and unbounded) for irrational timed automata. Thus, our algorithm for time bounded reachability will be presented for irrational timed automata.

For any initialized linear inclusion automaton \mathcal{A} with size n , the first step of our algorithm is to transform \mathcal{A} into a irrational timed automaton \mathcal{D} with size at most $2^{O(n \lg n)}$ such that \mathcal{D} has the same (bounded as well as unbounded time) reachability information. The translation from an initialized linear inclusion automaton \mathcal{A} to a irrational timed automaton \mathcal{D} is done in three steps. First an initialized linear equation automaton \mathcal{B} is created, wherein possible flows of variables are always deterministic. We then construct a stopwatch automaton \mathcal{C} from \mathcal{B} by replacing each variable x in $\mathbf{X}_{\mathcal{B}}$ by a clock t_x in $\mathbf{X}_{\mathcal{C}}$ and updating constraints appropriately. Constructing a timed automaton \mathcal{D} from \mathcal{C} wherein all the reachability information of \mathcal{C} is preserved, is the same as the classical algorithm described in [2]; therefore, this last step is not described here ².

To illustrate our construction, we use as a running example throughout this section, the initialized linear inclusion automaton of a thermostat shown in Figure 3.3. The automaton has two locations (1 and 2) and one variable x . Location 1 is the initial location (denoted by an incoming arrow with no source). Invariant and flows of x in each location are written inside that location. There are two edges between these two locations that are taken when $x = 3$ and when $x = 4$, respectively. Whenever a transition is taken, the variable x is reset to ensure that the automaton is initialized.

We call a linear inclusion automaton \mathcal{A} *normal* if it has the following properties:

²Recall that being a stopwatch automaton allows variables of \mathcal{C} to have the 0 flow in some locations, whereas variables in the timed automaton \mathcal{D} must always have 1 as their flows. Also in stopwatch automaton we can initialize and reset variables to some values other than zero, whereas in timed automaton we should always initialize and reset to zero.

- $\forall e : \mathbf{E}, x : \mathbf{X} \bullet \mathbf{G}(e, x) \subseteq \mathbf{I}(\mathbf{S}e, x)$
- $\forall e : \mathbf{E}, x : \mathbf{J}e \bullet \mathbf{R}(e, x) \subseteq \mathbf{I}(\mathbf{D}e, x)$
- $\forall e : \mathbf{E}, x : (\mathbf{X} \setminus \mathbf{J}e) \bullet \mathbf{G}(e, x) \subseteq \mathbf{I}(\mathbf{D}e, x)$

It is easy to see that for every linear inclusion automaton \mathcal{A} we can construct another linear inclusion automaton \mathcal{B} such that \mathcal{B} is normal, has the same size as \mathcal{A} , and preserves all the reachability information in \mathcal{A} . Therefore, in the rest of this section all automata are assumed to be normal (note that initialized linear inclusion automaton in Figure 3.3 is already normal).

3.2.1 From Linear Inclusion to Linear Equation

Given an initialized linear inclusion automaton \mathcal{A} , we want to construct an initialized linear equation automaton \mathcal{B} in which all the reachability information is preserved. We use the same translation that is used for initialized rectangular automata in which each variable x is replaced by two variables u_x and l_x to track its extremal values. The translation for initialized rectangular automata relies on two important properties that we need to ensure hold in the more general case of initialized linear inclusion automaton. First, for every location $q : \mathbf{Q}_{\mathcal{A}}$, variable $x : \mathbf{X}_{\mathcal{A}}$, and $r : \mathbf{I}_{\mathcal{A}}(q, x)$, we need $\mathbf{F}_{\mathcal{A}}(q, x)(r) \neq \emptyset$. This ensures that all reachable states have at least one valid flow which holds trivially in [2]. Next, l_x and u_x are initially set to be lower and upper bounds of the initial values of x . We need to prove that the reachable values of x after any time $t : \mathbb{R}_{\geq 0}$ is exactly the interval given by the value reached by l_x and u_x after time t .

In order to guarantee the first property, we first transform \mathcal{A} to an initialized linear inclusion automaton \mathcal{A}' such that \mathcal{A}' preserves all the reachability information in \mathcal{A} and for all locations $q : \mathbf{Q}_{\mathcal{A}'}$, variables $x : \mathbf{X}_{\mathcal{A}'}$, and $r : \mathbf{I}_{\mathcal{A}'}(q, x)$ we have either $\mathbf{F}_{\mathcal{A}'}(q, x)(r) \neq \emptyset$ or no matter what $\mathbf{F}_{\mathcal{A}'}(q, x)$ is, no time can be spent in q (therefore possible flows are not important anymore). The construction is as follows: For every location $q : \mathbf{Q}_{\mathcal{A}}$ we have two locations $q, q' : \mathbf{Q}_{\mathcal{A}'}$. Also $q \in \mathbf{Q}_{\mathcal{A}}^{\text{init}}$ iff $q \in \mathbf{Q}_{\mathcal{A}'}^{\text{init}} \wedge q' \in \mathbf{Q}_{\mathcal{A}'}^{\text{init}}$. Automaton \mathcal{A}' has an additional fresh variable z ($\mathbf{X}_{\mathcal{A}'} = \mathbf{X}_{\mathcal{A}} \cup \{z\}$) such that the following conditions are true for it:

- $\forall q : \mathbf{Q}_{\mathcal{A}'} \bullet \mathbf{F}_{\mathcal{A}'}(q, z) = 1 \leq \dot{x} \leq 1$, meaning z is a clock in \mathcal{A}' .

- $\forall q: \mathbb{Q}_{\mathcal{A}'} \bullet \mathbf{X}_{\mathcal{A}'}^{\text{init}}(q, z) = [0, 0]$, meaning the initial value of z is always 0.
- $\forall e: \mathbf{E}_{\mathcal{A}'} \bullet \mathbf{G}_{\mathcal{A}'}(e, z) = (-\infty, \infty) \wedge z \in \mathbf{J}_{\mathcal{A}'}e \wedge \mathbf{R}_{\mathcal{A}'}(e, z) = [0, 0]$, meaning satisfaction of guard does not depend on the value of z and is always reset to 0 on edges.
- $\forall q: \mathbb{Q}_{\mathcal{A}} \bullet \mathbf{I}_{\mathcal{A}'}(q, z) = [0, 0] \wedge \mathbf{I}_{\mathcal{A}'}(q', z) = (-\infty, \infty)$, meaning in the new locations, satisfaction of invariant does not depend on value of z , but in the original locations it prevents any time transition from happening.

Flows of variables other than z in \mathcal{A}' are the same as their flows in the original locations in \mathcal{A} . For each edge $e: \mathbf{E}_{\mathcal{A}}$ from location $q_1: \mathbb{Q}_{\mathcal{A}}$ to location $q_2: \mathbb{Q}_{\mathcal{A}}$, we have four edges in $\mathbf{E}_{\mathcal{A}'}$ from locations q_1, q_1, q'_1 and q'_1 , to locations q_2, q'_2, q_2 and q'_2 respectively (other elements of e are just updated to consider z as already specified). Finally, for each location $q: \mathbb{Q}_{\mathcal{A}}$ and variable $x: \mathbb{Q}_{\mathcal{A}}$, we have $\mathbf{I}_{\mathcal{A}'}(q', x) = \mathbf{I}_{\mathcal{A}}(q, x) \cap \{r: \mathbb{R} \cup (-\infty, \infty) \mid \mathbf{F}(q, x)(r) \neq \emptyset\}$ and $\mathbf{I}_{\mathcal{A}'}(q, x) = \mathbf{I}_{\mathcal{A}}(q, x) \setminus \mathbf{I}_{\mathcal{A}'}(q', x)$. Intuitively this means that invariants of variables in q' are restricted such that for all variable values that are allowed by the invariants we have at least one valid flow. Conversely, invariants of variables in q are restricted such that for all variable values that are allowed by invariants we have no valid flow (note that $\mathbf{I}_{\mathcal{A}'}(q', x)$ and $\mathbf{I}_{\mathcal{A}'}(q, x)$ are still intervals).

It is not hard to see that all the reachability information of \mathcal{A} is preserved by \mathcal{A}' . Furthermore, \mathcal{A}' by construction guarantees the first property. Note that \mathcal{A}' may not be normal anymore, so one may need to do an additional transformation to obtain a normal automaton \mathcal{A}'' . Figure 3.4 shows the initialized linear inclusion automaton \mathcal{A}' obtained from automaton \mathcal{A} in Figure 3.3. Location 1 in $\mathbb{Q}_{\mathcal{A}}$ is divided into locations 1 and 4 in $\mathbb{Q}_{\mathcal{A}'}$. Similarly location 2 in $\mathbb{Q}_{\mathcal{A}}$ is divided into locations 2 and 3 in $\mathbb{Q}_{\mathcal{A}'}$. A fresh variable z is added into the set of variables ($\mathbf{X}_{\mathcal{A}'} = \{x, z\}$) which is initially 0, is reset on all edges, puts no constraint on guards, and its flow is always 1. Locations $1, 4: \mathbb{Q}_{\mathcal{A}'}$ are initial locations and x is set to 3 in both. For each edge in $\mathbf{E}_{\mathcal{A}}$ we have 4 edges in $\mathbf{E}_{\mathcal{A}'}$ in which only source and destination locations are changed. The invariant $z = 0$ does not allow any time transition in locations 2 and 4. Furthermore, invariant $1 \leq x \leq 4$ in $2: \mathbb{Q}_{\mathcal{A}}$ is changed to $1 \leq x \leq 2$ in location $2: \mathbb{Q}_{\mathcal{A}'}$ and $2 \leq x \leq 4$ in location $3: \mathbb{Q}_{\mathcal{A}'}$. That is because when x is between 1 and 2, the lower bound of \dot{x} is larger than its upper bound

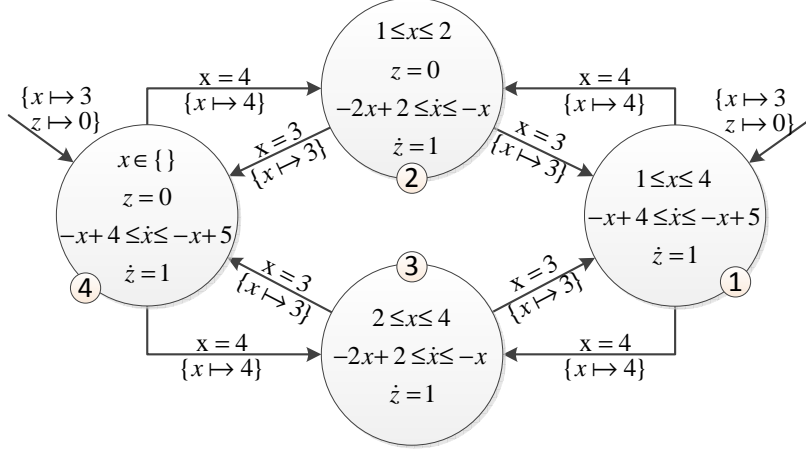


Figure 3.4: Example Initialized Linear Inclusion Automaton \mathcal{A}'

($-2x + 2 > -x$). Note that at $x = 2$ we have $-2x + 2 = -x$, so based on the construction method of \mathcal{A}' we should set invariant of x in $2:\mathbb{Q}_{\mathcal{B}}$ to $1 \leq x < 2$. We use non-strict inequality in order to have the same number of locations in the next step (the construction method multiplies the number of locations by $4^{|X|}$ if we have strict as well as non-strict constraints). It is easy to see that considering this special case does not change the reachability information. Also we know that $-x + 4$ is always smaller than $-x + 5$. Therefore the invariant of x in location $4:\mathbb{Q}_{\mathcal{A}'}$ is not satisfiable and we can remove this location from the automaton \mathcal{A}' (by normalizing it).

Recall that the second property says the reachable values of x after any time $t:\mathbb{R}_{\geq 0}$ is exactly the interval given by the value reached by variables l_x and u_x after time t . We prove this in Lemma 10. This generalizes the previous result for initialized rectangular automata.

Lemma 10. Let \mathcal{A} be an initialized linear inclusion automaton, q be any location of \mathcal{A} , and $t:\mathbb{R}_{\geq 0}$. Let $s:X \rightarrow \mathcal{I}$ be a set of valuations satisfying the invariant for q , i.e., for all x , $s(x) \subseteq \mathcal{I}(q, x)$. Let $\text{reach}_q(s, t)$ be the set of valuations reachable from s at time t . In other words, $\text{reach}_q(s, t) = \{\nu_2 \mid \exists \nu_1 \forall x:X. \nu_1(x) \in s(x) \wedge (q, \nu_1) \xrightarrow{t} (q, \nu_2)\}$. For a variable x if $\mathbf{F}(q, x) =$

($ax + b \leq \dot{x} \leq cx + d$) define $l_s^x(t)$ and $u_s^x(t)$ to be

$$l_s^x(t) = \begin{cases} l_{s(x)}e^{at} + \frac{b(e^{at} - 1)}{a} & \text{if } a \neq 0 \\ l_{s(x)} + bt & \text{otherwise} \end{cases}$$

$$u_s^x(t) = \begin{cases} u_{s(x)}e^{ct} + \frac{d(e^{ct} - 1)}{c} & \text{if } c \neq 0 \\ u_{s(x)} + dt & \text{otherwise} \end{cases}$$

$l_s^x(t)$ ($u_s^x(t)$) is the lower bound (upper bound) on x at time t provided the automaton starts from s . Then $\text{reach}_q(s, t) = \{\nu \mid \forall x \bullet \nu(x) \in [l_s^x(t), u_s^x(t)] \cap \mathbf{I}(q, x)\}$ ³.

Proof. We first prove that no point $\leq l_s^x(t)$ is reachable (similar argument proves that no point $\geq u_s^x(t)$ is reachable). Because $(q, \nu_1) \xrightarrow{t} (q, \nu_2)$, for each variable $x : \mathbf{X}$ there is a differentiable function $f_x : [0, t] \rightarrow \mathbb{R}$ such that $\forall r : [0, t] \bullet \dot{f}_x(r) \in F(q, x)(r)$. Since f is differentiable it is continuous too, and since $\nu_1(x) \geq l_s^x(0) = l_{s(x)}$ if $\nu_2(x) < l_s^x(t)$ then $\exists t_c : [0, t] \bullet f_x(t_c) = l_s^x(t_c) \wedge f_x(t_c^+) < l_t^x(t_c^+)$. But this means $\dot{f}_x(t_c) < l_{F(q, x)(t_c)}$ which is a contradiction. Therefore no point outside $[l_s^x(t), u_s^x(t)] \cap \mathbf{I}(q, x)$ is reachable.

To prove that for all variable $x : \mathbf{X}$, all points p in $[l_s^x(t), u_s^x(t)] \cap \mathbf{I}(q, x)$ are reachable, first consider the case $\mathbf{I}(q, x) = (-\infty, \infty)$. In this case we know that both $l_s^x(t)$ and $u_s^x(t)$ are reachable and prove that their convex combinations are reachable too. If we take f_x to be $\lambda l_s^x + (1 - \lambda)u_s^x$ then it is easy to see that for all $r : [0, t]$, $f_x(r) : F(q, x)(r)$. To consider the case when $\mathbf{I}(q, x) \neq (-\infty, \infty)$ we first prove that $\min\{u_{\mathbf{I}(q, x)}, u_s^x(t)\}$ is reachable (similar argument proves that $\max\{l_{\mathbf{I}(q, x)}, l_s^x(t)\}$). Then using the previous argument it is easy to see that all points between this minimum and maximum are reachable. If $u_{\mathbf{I}(q, x)} > u_s^x(t)$ then the proof is obvious. Otherwise we need to note that if the invariant was $(-\infty, \infty)$, $u_{\mathbf{I}(q, x)}$ was a reachable point. And because f_x is a monotonic function, we know that f_x is always inside $\mathbf{I}(q, x)$. This proves that $u_{\mathbf{I}(q, x)}$ is reachable even if $\mathbf{I}(q, x) \neq (-\infty, \infty)$. \square

Figure 3.5 displays the initialized linear equation automaton \mathcal{B} obtained from the initialized linear inclusion automaton \mathcal{A}' in Figure 3.4 after removing location $4 : \mathbf{Q}_{\mathcal{A}'}$ (remember that invariant of that location was not satisfiable). It has the same set of locations. We replace $x : \mathbf{X}_{\mathcal{A}'}$ by two variables $x_l, x_u : \mathbf{X}_{\mathcal{B}}$

³open and unbounded regions are handled by additional bookkeeping exactly as it is done in [2].

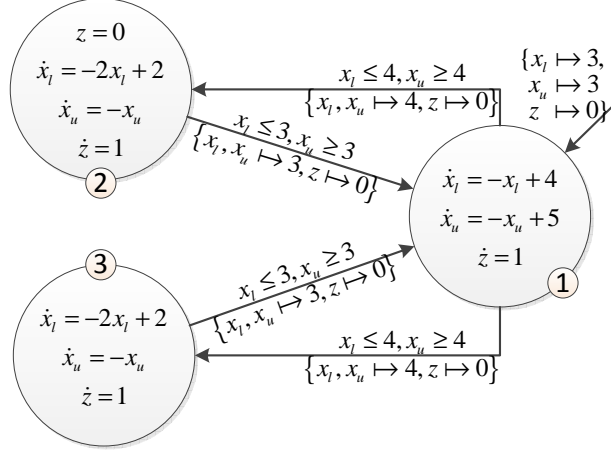


Figure 3.5: Example Initialized Linear Equation Automaton \mathcal{B}

that respectively track lower and upper bounds of $x : \mathbf{X}_{\mathcal{A}'}$ ⁴. Note that we do not replace $z : \mathbf{X}_{\mathcal{A}'}$ by two variables, because possible flows of z , its initial values, and its reset values are always singletons. In addition to the new variables, two other changes occur in \mathcal{B} . First, the guard $x = a$ (for $a = 3$ or $a = 4$) on edges of $\mathbf{E}_{\mathcal{A}'}$ is replaced by $x_l \leq a$ and $x_u \geq a$ on edges of $\mathbf{E}_{\mathcal{B}}$. Intuitively this checks that the reachable region has an intersection with the guard. Second, invariant of $x_l, x_u : \mathbf{X}_{\mathcal{B}}$ is $(-\infty, \infty)$ in locations of \mathcal{B} . This is because we want to let x_l and x_u to reach everywhere possible. Otherwise, it is possible for example that after 1 unit of time, x_l hits the lower bound but x_u does not hit the upper bound. But since x_l would not longer satisfy the invariant, no more time transition can occur. On the other hand, it is possible to assign \dot{x} the upper bound of possible flows and reach the upper bound of invariant. The invariants of \mathcal{A}' are considered in a function that maps each reachable state in $\mathbf{S}_{\llbracket \mathcal{B} \rrbracket}$ to a set of states in $\mathbf{S}_{\llbracket \mathcal{A}' \rrbracket}$. Note that we assume that guards are always subset of invariants. Therefore when the reachable region has an intersection with a guard, we know that the result of the intersection is always subset of invariants.

3.2.2 From Linear Equation to Stopwatch

For an initialized linear inclusion automaton \mathcal{A} , the initialized linear equation automaton \mathcal{B} obtained in Section 3.2.1, is initialized and has the following

⁴If there was any non-compact constraint in \mathcal{A}' then $\mathbf{X}_{\mathcal{B}}$ would have one more variable than $\mathbf{X}_{\mathcal{A}'}$.

features:

- $\forall q : \mathbb{Q}^{\text{init}}, x : \mathbf{X} \bullet |\mathbf{X}^{\text{init}}(q, x)| \leq 1$,
- $\forall e : \mathbf{E}, x : \mathbf{J}e \bullet |\mathbf{R}(q, x)| \leq 1$, and
- $\forall q : \mathbb{Q}, x : \mathbf{X} \bullet l_{\mathbf{F}(q, x)} = u_{\mathbf{F}(q, x)}$, where $l_{\mathbf{F}(q, x)}$ and $u_{\mathbf{F}(q, x)}$ are, respectively, lower and upper bounds of $\mathbf{F}(q, x)$.

Such an automaton is called *solvable* in [49]. Given a solvable hybrid automaton \mathcal{B} , we can use the clock translation method to construct an initialized stopwatch automaton \mathcal{C} that preserves all the reachability information in \mathcal{B} . In this method every variable $x : \mathbf{X}_{\mathcal{B}}$ is replaced by a clock that tracks the time elapsed since the last time x was reset. Using this time the value of x can be computed using the Equation 2.1 and this is used to change the constants on x . The reader is referred to [49] for an explanation of this method. Finally, the stopwatch automaton \mathcal{C} is translated to a timed automaton \mathcal{D} , in the standard way [49].

Remark 11. Constants in the final timed automaton are of the form r or $r \ln r'$ in which $r : \mathbb{Q}$ and $r' : \mathbb{Q}_+$. Note that in the case of $r \ln r'$, r is always $\frac{1}{a}$ for some $\dot{x} = ax + b$. The important point is that since \mathcal{A} is initialized, variables whose flow changes in a discrete step are reset. Therefore, in automaton \mathcal{D} , a variable cannot be compared with both r and $r \ln r'$ unless the variable is reset in between. Similarly a variable cannot be compared with both $r_1 \ln r'_1$ and $r_2 \ln r'_2$ such that $r_1 \neq r_2$ unless the variable is also reset in between.

Note that for every location q in $\mathbb{Q}_{\mathcal{D}}$ (same as $\mathbb{Q}_{\mathcal{C}}$) there exists one and only one corresponding location in $\mathbb{Q}_{\mathcal{B}}$. In other words, transformations from \mathcal{B} to \mathcal{D} may split locations, but they never merge them. Figure 3.6 displays the stopwatch automaton \mathcal{C} obtained from initialized linear equation automaton \mathcal{B} in Figure 3.5. Because in the initialized linear equation automaton no variable has a zero flow, \mathcal{C} is also a timed automaton. It has the same set of locations. Initial values of variables are always zero, and variables are always reset to zero on edges. We use the clock translation only for the variable $x_l, x_u : \mathbf{X}_{\mathcal{B}}$ because z is already a clock. Note that we end up with some logarithmic constants. Lemma 12 formalizes the results in this section so far, and proves a bound on the size of the irrational timed automaton \mathcal{D} .

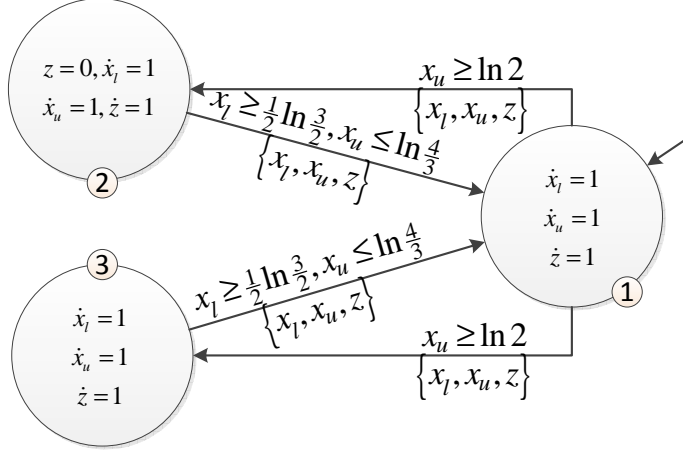


Figure 3.6: Example Stopwatch Automaton \mathcal{C}

Lemma 12. For any initialized linear inclusion automaton \mathcal{A} with size n , there is a irrational timed automaton \mathcal{D} with size at most $2^{O(n \lg n)}$ such that \mathcal{D} has the same (bounded as well as unbounded time) reachability information. More precisely, there is a function f that maps a subset of $\mathbb{S}_{[\mathcal{D}]}$ to a subset of $\mathbb{S}_{[\mathcal{A}]}$ such that if D is the set of reachable states in \mathcal{D} after bounded or unbounded amount of time, $f(D)$ is the set of reachable states in \mathcal{A} after the same amount of time.

Proof. Existence of \mathcal{D} and f are clear from construction. We only prove the statement about size of \mathcal{D} . Assume we denote the number of constants in an initialized linear inclusion automaton \mathcal{H} by $K_{\mathcal{H}}$. With regards to the number of variables, we have $|X_{\mathcal{D}}| = |X_{\mathcal{C}}| = |X_{\mathcal{B}}| \leq 2|X_{\mathcal{A}}| + 3$. Number of constants in automaton \mathcal{B} is equal to their number in automaton \mathcal{A}' and is at most $|Q_{\mathcal{A}}| \times |X_{\mathcal{A}}| \times K_{\mathcal{A}}$, because for each variable and each location in \mathcal{A} , only one invariant may change. Number of constants in automata \mathcal{C} and \mathcal{D} are equal and is at most $|X_{\mathcal{B}}| \times |Q_{\mathcal{A}}| \times K_{\mathcal{B}}$. $|X_{\mathcal{B}}| \times |Q_{\mathcal{A}}|$ is the maximum number of different flows in \mathcal{B} and solving \mathcal{B} using Equation 2.1 may replace each constant by at most $|X_{\mathcal{B}}| \times |Q_{\mathcal{A}}|$ number of them. Number of locations in automaton \mathcal{B} is at most $|Q_{\mathcal{A}}| \times 2^{|X_{\mathcal{A}}|} \times 2^{4|X_{\mathcal{A}}|+3} = |Q_{\mathcal{A}}| \times 2^{5|X_{\mathcal{A}}|+3}$ in which $|Q_{\mathcal{A}}| \times 2^{|X_{\mathcal{A}}|}$ is the maximum number of locations in \mathcal{A}' . Number of locations in automaton \mathcal{C} is at most $|Q_{\mathcal{B}}| \times K_{\mathcal{B}}^{|X_{\mathcal{B}}|}$. And number of locations in automaton \mathcal{D} is at most $|Q_{\mathcal{C}}| \times (K_{\mathcal{C}} + 1)^{|X_{\mathcal{C}}|}$. Therefore assuming \mathcal{A} has at most n variables, n locations, and n different constants, $|X_{\mathcal{D}}| \leq 2n + 3$, $K_{\mathcal{D}} \leq (2n + 3) \times n^4 = 2n^5 + 3$, and $|Q_{\mathcal{D}}| \leq F'(A) = n \times 2^{5n+3} \times n^{6n+9} \times (n^3 + 1)^{2n+3} \in 2^{O(n \lg n)}$. Note that $K_{\mathcal{D}}$ is

at most polynomially larger than $K_{\mathcal{A}}$ and $Q_{\mathcal{D}}$ is at most *single* exponentially larger than $|Q_{\mathcal{A}}|$. Finally, assuming each constant in \mathcal{A} is represented as a pair of n -bit integers, each constant in \mathcal{A}' can be represented by a pair of at most $4n + 1$ bits. Size of constants in automata \mathcal{A}' and \mathcal{B} and in automata \mathcal{C} and \mathcal{D} are equal. Constants in \mathcal{C} are in the form of r_1 or $r_2 \ln r_3$ wherein using Equation 2.1 we know

1. each element of r_1 has at most $5n + 1$ bits,
2. each element of r_2 has at most n bits, and
3. each element of r_3 has at most $10n + 3$ bits.

□

3.3 Bounding the Execution Length in Irrational Timed Automata

The constants in the irrational timed automaton \mathcal{D} constructed by Lemma 12 are not rational. Thus, we cannot use the well known techniques for analyzing rational timed automata [11]. Our decidability proof relies on first observing that if a configuration (q, ν) is reachable in \mathcal{D} within bounded time T , it is reachable by an execution of bounded length. Our algorithm therefore guesses an execution of this length and, by solving some constraints (Section 3.3.3), the algorithm checks if it is a valid execution that starts from some initial state and ends in some unsafe state. Finally, we show that the problem is PSPACE-hard in Section 3.3.4.

In this section we bound length of the execution that we need to guess. Our proof closely follows the proof outlined in [16] for monotonic rectangular automata. Notice that a irrational timed automaton is a special monotonic rectangular automaton with the difference that it may have irrational constants in its constraints. The presence of irrational constants introduces challenges that we address in this section. We begin by giving a short outline of the proof in [16] to highlight the key challenges that will need to be addressed when considering irrational timed automata. We then present our proof.

3.3.1 Brihaye *et al.*'s Algorithm for Monotonic Rectangular Automata

The main observation in [16] is that if there is a run ρ of monotonic rectangular automaton \mathcal{D} from state (q_1, ν_1) to (q_2, ν_2) such that $\text{duration}(\rho) \leq T$ then there is a run ρ' from (q_1, ν_1) to (q_2, ν_2) of the same duration, whose length is exponential in the size of \mathcal{D} and linear in T . The construction of ρ' from ρ relies on a *contraction operator*. The contraction operator identifies positions $i < j$ in ρ that have the same location such that all the locations between i and j are also visited before i in ρ . The operator then deletes all the locations $i + 1, \dots, j$ and adds their time to the previous occurrences before i . Brihaye *et al.* apply this operator as many times as required until a fix-point is reached. They show that the resulting run (after contraction) has at most $|\mathbb{Q}_{\mathcal{D}}|^2 + 1$ edges. The problem of course is that the run after contraction may no longer be a valid run. The contracted run would be valid only if the run to which we apply contraction has some special properties. Brihaye *et al.*, therefore, first partition the run ρ carefully into exponentially many fragments such that the contraction operator can be reliably applied to these fragments, and the resulting run is a genuine run.

We now describe how they partition the run into fragments to which the contraction operator can be applied. Observe that since the contraction operator removes certain locations from the run by adding the time spent in these locations to the time spent in the same locations earlier in the run, such an operation can be sound only if the valuations in the merged locations satisfy the same set of constraints. If valuations in the merged locations don't satisfy the same set of constraints then the invariants of locations and guards of transitions may be violated by the merging process. Thus, Brihaye *et al.* first transform the automaton \mathcal{D} into an automaton \mathcal{E} that keeps track of the “region” of the valuation in its control state. Let \mathcal{D} be an automaton with positive rates such that all constants appearing in the constraints are natural numbers⁵ and cmax is the maximum constant in \mathcal{D} . A region r (according to [16]) is a set of valuations that satisfy the same set of constraints of the form $x \triangleleft c$, where x is a variable of \mathcal{D} , $\triangleleft: \{<, \leq\}$ and c is a natural number $\leq \text{cmax}$; thus a region r is similar to region of a timed automaton, except that the order of the fractional values is not maintained. The automaton \mathcal{E}

⁵Any automaton with rational constants can be transformed into such a machine

has locations of the form (q, r) where q is a location of \mathcal{D} and r is a region, such that all its runs are *region consistent*. A run $\rho = ((q_0, r_0), \nu_0), (t_1, e_1), ((q_1, r_1), \nu_1), \dots, (t_n, e_n), ((q_n, r_n), \nu_n)$ is region consistent if $\nu_i \in r_i$ for all i . In addition, for variables x that enter a location with value 0, \mathcal{E} keeps track of whether the value x becomes > 0 before the next transition, or it never changes from 0 before the next transition. This is required to bound the number of sub-runs that are constructed later, and prevents the contraction operator from merging states where x stays 0 with those where x becomes > 0 . The construction ensures that \mathcal{D} admits a run between two states of duration T iff \mathcal{E} admits a run between the same states and for the same duration T and length.

Let us consider an arbitrary T -bounded run ρ of \mathcal{E} . Assuming that each location in \mathcal{E} has a self loop that can be taken at anytime, one can construct an “equivalent” run ρ_0 that is the concatenation of at most $T \times \text{rmax} + 1$ shorter runs, each of duration at most $\frac{1}{\text{rmax}}$; these shorter runs are called type-1 runs. If rmax is taken to be the maximum rate of flow of any variable in \mathcal{E} , then a type-1 run has the property that any variable changes its non-zero region at most 3 times within that run, because within $\frac{1}{\text{rmax}}$ time, no variable can change its value by more than 1. Splitting each type-1 run at the points when a variable changes its non-zero region, results in $3 \times |\mathbf{X}_{\mathcal{E}}|$ type-2 runs, where variables with value ≥ 1 never change their region. The only change in regions involves variables whose value changes from 0 to a value in $(0, 1)$, and there could be unbounded number of region changes of this form. The contraction operator when applied to a type-2 results in a valid run, but the problem is that the valuation in the last state can be different after contraction. Changing the last state of run does not allow one to concatenate all the contracted type-2 runs to get a valid run of \mathcal{E} . To address this problem each type-2 run is subdivided into type-3 runs based on when a variable was first and last reset within a type-2 run. Applying the contraction operator to type-3 runs, and then concatenating them back, results in a valid contracted type-2 run of the same duration, with the same starting and ending states. These contracted type-2 runs are then concatenated back to get a run ρ' that is bounded length, has the same duration as ρ , and has the same start and end states. Having established that \mathcal{E} has T -bounded runs iff \mathcal{E} has T -bounded runs of length at most $F(\mathcal{D}, T) = 24 \times (T \times \text{rmax} + 1) \times |\mathbf{X}_{\mathcal{D}}|^2 \times |\mathbf{Q}_{\mathcal{D}}|^2 \times (2 \times \text{cmax} + 3)^{2|\mathbf{X}_{\mathcal{D}}|}$, to solve time bounded reachability, the NEXPTIME algorithm

non-deterministically guesses a run of length at most $F(\mathcal{D}, T)$ and solves a linear program to check if there are time points and valuations for each step that make the run feasible.

3.3.2 Time-Bounded Reachability for Irrational Timed Automata

The algorithm described in [16] (and outlined in above) cannot be directly applied to irrational timed automata due to the presence of irrational constants. There are 2 main challenges we need to address.

1. For the contraction operator to be correctly applied, we need a way to partition valuations into finitely many regions that ensures that all valuations in a region satisfy the same constraints. When irrational constants appear in constraints, we can no longer define regions based on how the values of variables compare to a finite set of natural numbers. Instead we need a new definition of regions.
2. Type-1 runs are runs of short duration, where there are only a bounded number of certain types of region changes. The duration for such a type-1 has to be such that any run can be divided into at most exponentially many type-1 runs. We will need to identify what the right duration of a type-1 run should be given the changed definition of regions.

We solve each of these challenges in order. We begin by describing a new definition of regions. For any variable x , let $\text{cnst}(x) \subset \mathbb{R}$ be the set of real constants used as an interval end-point for constraints over x (this includes initial values, invariants, guards, and resets). Let $\text{cnst}^+(x)$ be the set of positive values in $\text{cnst}(x)$, and $\text{cnst}^{\max}(x)$ be the maximum element in the non-empty set $\text{cnst}(x)$.

Definition 13. Given a irrational timed automaton \mathcal{D} , for each variable $x : X$ we define $\text{reg}(x)$ as the set of intervals created by the constants used in constraints of x .

$$\begin{aligned} \text{reg}(x) = & \{0^-, 0^+, (\text{cnst}^{\max}(x), \infty)\} & \cup \\ & \{[c, c] \mid c : \text{cnst}^+(x)\} & \cup \\ & \{(a, b) \mid a, b : \text{cnst}(x) \wedge \forall c : \text{cnst}(x) \bullet c \notin (a, b)\} \end{aligned}$$

$\text{Region}(D)$ is the set of all functions that map each variable $x : \mathbf{X}$ to an element of $\text{reg}(x)$. A region $r : \text{Region}(D)$ can be thought of as the set of valuations ν such that for every variable x , $\nu(x) \in r(x)$ (where $\nu(x) \in 0^-$ and $\nu(x) \in 0^+$ iff $\nu(x) = 0$). We will interchangeably think of regions in this way.

Just like [16], in the definition of regions, we distinguish between the case when a variable is 0 and no time will be spent before the next transition (0^-), and when a variable is 0 but some non-zero time is promised to be spent before the next transition (0^+). The region definition above is different from [16] in that we only compare the value of variable to the constants that appear in the constraints, as opposed to all numbers upto some maximum bound. Using this new definition of regions, given a irrational timed automaton \mathcal{D} , we construct automaton \mathcal{E} that remembers the region of the valuation when a run enters a location.

The second challenge pertains to determining the duration of type-1 runs (and hence type-2 runs). In [16] the duration is picked to ensure two properties:

1. any run can be divided into an exponential number of type-2 runs, where the regions of variables don't change;
2. in a type-2 run, since the region of a variable does not change, all valuations in a type-2 run satisfy the same set of constraints, and so contracting the run results in valid run of \mathcal{E} ⁶. Now, we could pick the duration of a type-1 run such that in any such run there are at most 3 changes to the region of any variable, by estimating the distance between any two constants that define regions. However, any estimation of the closest distance between two constants (of the form $r_1 \ln r_2$ and $r_3 \ln r_4$) appearing in a irrational timed automaton yields a doubly-exponential bound. This prevents us from exponentially bounding the number of type-2 runs.

Instead, we relax condition 2 when picking our duration to ensure that there are only exponentially many type-2 runs. Observe that in irrational timed automata, by definition, a variable cannot be compared with constants

⁶In addition we would like the contracted run to start and end in the same state. But this is easily accomplished by splitting type-2 runs into type-3 runs, and applying the contraction to type-3 runs.

of different types (q_1 and $q_2 \ln q_3$, or $q_1 \ln q_2$ and $q_3 \ln q_4$ for $q_1 \neq q_3$) without being reset in between. Therefore, instead of requiring that type-2 runs consist of valuations belonging to the same region, we define type-2 runs to be ones where the flows of the variables (in the original initialized linear inclusion automaton \mathcal{A}) remain the same, and all valuations satisfy the same set of *relevant* constraints. Thus, even though two valuations in a type-2 run may satisfy different set of constraints (and therefore belong to different regions), since they satisfy the same constraints that would pertain to them (without a reset), applying the contraction operation will yield a valid run. We, therefore, pick the duration of a type-1 run to be determined by the minimum distance between constants of the form r_1 and r_2 , where r_1 and r_2 are rationals, or between constants of the form $r \ln r_1$ and $r \ln r_2$, where r, r_1 , and r_2 are rational numbers. Considering constant sizes given at the end of proof of Lemma 12, we have three cases:

- Both constants are rationals. In this case, the difference between two distinct constants cannot be smaller than $\frac{1}{2^{10n+2}}$.
- Both constants are logarithmic. In this case, if constants are in the form of $r \ln r_1$ and $r \ln r_2$ then $\ln \frac{r_1}{r_2}$ cannot be smaller than $\frac{1}{2^{20n+7}}$. When this number is multiplied by r it cannot be smaller than $\frac{1}{2^{21n+7}}$. Note that as a special case, $r \ln r_2 = 0$, the distance between $r \ln r_1$ and 0 cannot be smaller than $\frac{1}{2^{11n+4}}$.
- One constant is rational and the other one is logarithmic. We only need to consider the case when the rational number is 0, which is special case of the previous bullet.

Taking the duration of a type-1 run to be at most $\frac{1}{2^{21n+7}} = \min\{\frac{1}{2^{10n+2}}, \frac{1}{2^{21n+7}}\}$, the set of relevant and satisfied constraints is changed at most 3 times for any variable x . Now, any T -bounded run can be divided into at most $T \times 2^{21n+7} + 1$ type-1 runs. This concludes that $F''(\mathcal{D}, T)$ the bound on length of the run that is needed to be guessed is $24 \times (T \times 2^{21n+7} + 1) \times |\mathbf{X}_{\mathcal{D}}|^2 \times |\mathbf{Q}_{\mathcal{D}}|^2 \times (2K_{\mathcal{D}} + 3)^{2|\mathbf{X}_{\mathcal{D}}|}$. Therefore $F''(\mathcal{D}, T) \leq 24 \times (T \times 2^{21n+7} + 1) \times (2n + 3)^2 \times 2^{10n+6} \times n^{12n+20} \times (n^3 + 1)^{4n+6} \times (2n^5 + 3)^{2n+6} \in 2^{O(n \ln n)}$.

3.3.3 Algorithm for Time-Bounded Reachability

Since a reachable configuration is reachable by bounded length execution, the algorithm for time bounded reachability will guess an execution of appropriate length (using polynomial space), and check if the guessed execution is valid. Note that checking the validity of a guessed execution involves constraints with transcendental numbers, and hence cannot be solved using linear programming. We outline how this can be carried out.

Our path validity constraints are difference constraints of a special form. As has been observed in Theorem 3 in [54], checking feasibility of these constraints can be reduced to checking for the existence of negative cost cycles in a weighted directed graph; this graph has as many vertices as the length of run whose feasibility we are checking, and the weights are the constants appearing in the timed automaton. Alur *et al.*, in [54], presented a modified shortest path algorithm that checks the feasibility of these constraints and runs in time $O(|X|^2|\pi|)$, where π is the guessed run, and uses space only $O(|X|)$. Thus, the space requirements are only polynomial in the size of the automaton and is independent of the length of the execution π . However, since this algorithm computes costs of paths, it involves adding weights and comparing them. To complete the description, we need to show how one can compare the costs arising during such a computation. The challenge involves comparing numbers involving natural logarithms. The algorithm in [54] only looks for simple cycles, and thus never adds the weight of an edge more than once in any of the values it computes. Thus, in the worst case, the algorithm requires comparing 0 with the sum of exponentially many constants. Observe that we have at most $O(n)$ distinct constants (n is bound on the size of the input automaton). Thus, the comparisons the algorithm needs to perform will be of the form $\sum_{0 \leq i < n} a_i \ln \frac{b_i}{c_i} < \sum_{0 \leq i < n} d_i$ where a_i, b_i, c_i, d_i are integers of $O(n)$ bits. There are many algorithms that efficiently compute natural logarithms with arbitrary precision; one can compute k bits of a number that approximates $\ln \frac{b}{c}$ (where b and c are n -bit integers) with error at most 2^{-k} using space that is polynomial in n and logarithmic in k by combining ideas in [50, 51, 55]. However, we are unaware of any complexity bounds on computing the k^{th} bit of $\ln \frac{b}{c}$, except in special cases like $\ln 2$ [56]. If the k^{th} bit of a linear combination logarithms of n -bit rationals can be computed in PSPACE, then we can bound the complexity of our algorithm to PSPACE,

because we only need to compute $O(n)$ bits of the left-hand side, since the right hand side is an integer with $O(n)$ bits.

In the absence of an algorithm to compute bits of natural log, we observe that the left hand side can never be equal to the right hand side, since the left hand side is an irrational number while the right hand side is an integer. Thus, our algorithm will compute the left hand side with increasing precision using either ideas from [50, 51, 55] or [52]. If the precision of our approximation is 2^{-k} and the difference between our approximation of the left hand side and right hand side is $> 2^{-k}$, then we can be sure of whether the inequality holds or not. Since the left hand side is not equal to the right hand side, we are guaranteed that eventually this will happen, giving us our decidability result.

We can prove the time bounded reachability problem is PSPACE-hard (see Section 3.3.4). The NEXPTIME-hardness lower bound from [15, 16] does not seem to extend to this case. Intuitively, the main reason that in the simulation of a 2-counter machine, monotonic rectangular hybrid automata can multiply a counter by a constant in a constant amount of time. On the other hand, because our machines are initialized, this does not seem possible for initialized linear inclusion automata. Though we cannot give complexity bounds on our algorithm because of difficulties in bounding the complexity of computing natural logs, we conjecture that this problem is PSPACE-complete.

3.3.4 Time-Bounded Reachability is PSPACE-hard in Initialized Linear Inclusion Automata

Recall that the reachability problem (without time bound) is PSPACE-hard [11] for rational timed automata. The reduction described in [11] reduces the halting problem of a linear bounded automaton to the reachability problem of timed automata, where each step of the linear bounded automaton is simulated in fixed time τ by the timed automaton. Now, recall that a linear bounded automaton halts iff it halts in $N = |Q| \times (n + 1) \times |L|^n$ steps, where $|Q|$ is the number of states of the linear bounded automaton, $|L|$ is the size of its tape alphabet, and n is the size of the input to the linear bounded automaton. Thus, the linear bounded automaton halts iff the constructed timed automaton reaches the desired control location within $T = N \times \tau$ time.

Since $N \times \tau = O(2^n)$, we can write T in polynomial time.

3.4 Related Work

The decidability/undecidability boundary for the reachability problem in hybrid automata has been delineated through a collection of results; the main results are included in the following references [2–12, 57]. Given that the reachability problem is in general undecidable, approximations to the reachability problem (and the reachable set of states) have been introduced [58–60]. The time bounded reachability problem has been shown to be decidable for monotonic rectangular automata in [15, 16] and for semi-algebraic o-minimal systems [61]. However, these classes of automata are incomparable to the class considered in this chapter. Semi-algebraic o-minimal systems [61] require the continuous variables to be reset on every discrete transition (or on every “cycle” of transitions), thus decoupling the discrete and continuous dynamics. Such a requirement is not imposed on our automata. Detailed comparison between monotonic rectangular automata and the automata considered here, is presented in the introduction and Section 3.3.2.

3.5 Conclusions

In this chapter, we considered initialized hybrid automata whose flows are described by linear differential inclusions, and whose invariants, guards, and resets are rectangular constraints. These results are published in [62, 63]. Such automata generalize both initialized rectangular automata and timed automata. We proved that the reachability problem (when time is not bounded) is undecidable, while the time bounded reachability problem is decidable. The only reason why we cannot obtain a complexity bound on our algorithm is because we are unaware of any complexity bounds on computing the i^{th} bit a natural logarithm of a rational number (except in special cases). There are few open problems left by our investigations. The most interesting is obtaining complexity bounds on computing natural logarithms of rational numbers. Another question is whether the results in [16] can be extended to monotonic linear inclusion automata.

Chapter 4

Approximate Model Checking

The main drawback of our decidability result in Chapter 3, is that the class of automata it applies to, is rather restrictive. In other words, for most systems one cannot use the decision procedure we gave in Chapter 3. Knowing that safety model checking is undecidable for general systems, means that one cannot hope to solve this problem for complex automata. A remedy to this problem involves crafting an abstract model with simpler dynamics that is amenable to automated analysis. The success of the abstraction based method depends on finding the right abstraction, which can be difficult. One approach that tries to address this issue is the Counter-Example Guided Abstraction Refinement (CEGAR) technique [17] that tries to automatically discover the right abstraction through a process of progressive refinement based on analyzing spurious counter-examples in abstract models. CEGAR has been found to be useful in several contexts [18–21], including hybrid systems [22–29].

There are two principal CEGAR approaches in the context of verifying hybrid automata that differ primarily on the space abstract models considered. The first approach [22–25, 28, 45] tries to abstract hybrid models into finite state, discrete transition systems that have no continuous dynamics. The second approach [27, 29, 64] abstracts a hybrid automaton by another hybrid automaton with simpler dynamics. Using hybrid automata as abstractions has the advantage that constructing abstract models is computationally easier.

In this chapter, we present a CEGAR framework for verifying cyber-physical systems, where the concrete and abstract models are both hybrid automata. We consider both non-linear polyhedral automata and affine hybrid automata here. In both classes, initial states, invariants, and reset relations are specified using polyhedral constraints. The difference is in the specification of their dynamics. While in affine hybrid automata, flows are

specified using only affine combinations of variables, in non-linear polyhedral automata flows could also be non-linear functions¹ of variables (see Section 2.2.1 for precise definitions). The safety verification problem for such hybrid automata is challenging — not only is the problem undecidable, but even for the case of affine dynamics, it is unknown whether the problem of checking if the states reachable within a time bound T (without taking any discrete transitions) intersects a polyhedral unsafe region is decidable. We abstract both affine hybrid automata and non-linear automata by polyhedral automata. Polyhedral automata are same as affine hybrid automata and non-linear automata, except that flows in polyhedral automata are also specified using polyhedral constraints. Our results extend earlier hybrid automata based CEGAR algorithms [27, 29, 64, 65] to a richer class of hybrid models (from concrete automata that have rectangular dynamics to automata that have affine dynamics or even non-linear dynamics).

We establish a few basic results about our CEGAR framework. First, we show that any spurious counter-example can be detected during the counter-example validation step. This result is not obvious because, as we mentioned, it is unknown whether the bounded time reachability problem is decidable even for affine hybrid automata. Hence validation cannot be carried out “exactly”. Our proof relies on the observation that the sets computed during counter-example validation are bounded, and uses the fact that continuous time bounded posts of affine hybrid automata can be approximated with arbitrary precision. Arbitrary approximation of non-linear dynamics in non-linear polyhedral automata is a non-trivial result and we dedicate Section 4.2 to it. Next, we show that our refinement algorithm makes progress. More precisely, we prove that any abstract counter-example, if it appears sufficiently many times, is eventually eliminated. Progress for the case of affine dynamics is proved by observing that, for a bounded time, linear dynamics can be approximated with arbitrary precision by rectangular dynamics [66]. Progress in the case of non-linear dynamics is because of arbitrary approximation of non-linear dynamics that we prove in Section 4.2.

We have extended our CEGAR-based tool **HARE** (Hybrid Abstraction Refinement Engine) once to verify affine hybrid automata, and another time to verify non-linear polyhedral automata; the first version of **HARE** only handled

¹We support arithmetic, trigonometric, and exponential functions.

rectangular automata. Furthermore, we found existing tools for model checking even rectangular automata (HyTech [67], PHAVer [43], SpaceEx [42], and FLOW* [68]) inadequate for our purposes (see Section 4.4 for explanations). So we implemented a new model checker for polyhedral automata that uses the Parma Polyhedral Library (PPL) [69] and Z3 [70]. Refinement uses IAL [71] and PPL libraries, and counter-example validation is carried out using PPL and by making calls to dReach [40].

For the case of affine dynamics, we have compared the performance of the new version of HARE against SpaceEx and PHAVer (more precisely with SpaceEx with the Supp and PHAVer scenarios), and with SpaceEx AGAR [44]². SpaceEx is a state-of-the-art symbolic state space explorer for affine hybrid automata that over-approximates the reachable set, and may occasionally converge to a fixed-point in the process. SpaceEx AGAR is a CEGAR-based tool that merges different locations and over-approximates their dynamics. The running time of HARE was almost always better than SpaceEx, PHAVer, and SpaceEx AGAR (details in Section 4.4). We also found that HARE was more *accurate*. On quite a few examples, SpaceEx and SpaceEx AGAR fail to prove safety either because they do not converge to a fixed-point or because their over-approximate the reach set too much. For the case of non-linear dynamics, we have compared the performance of HARE against HSolver [45], C2E2 [46], and FLOW* [47]. A virtual machine for the new HARE, along with examples and scripts for running them can be downloaded from <https://uofi.box.com/v/HARE>.

Our CEGAR framework, algorithms for abstraction, counter-example validation, and refinement, that form the theoretical basis for HARE, are described in Section 4.1. Arbitrary approximation of reachable sets for non-linear dynamics is explained in Section 4.2, and the requirement for doing both backward and forward reachability is explained in Section 4.1.7. Finally, the tool architecture and its internals are presented in Section 4.3, and Section 4.4 reports our experimental results.

²We stopped comparing HARE with HSolver on affine hybrid automata, since our experiments in [65] showed that HSolver can only handle one example out of more than 60 examples we had.

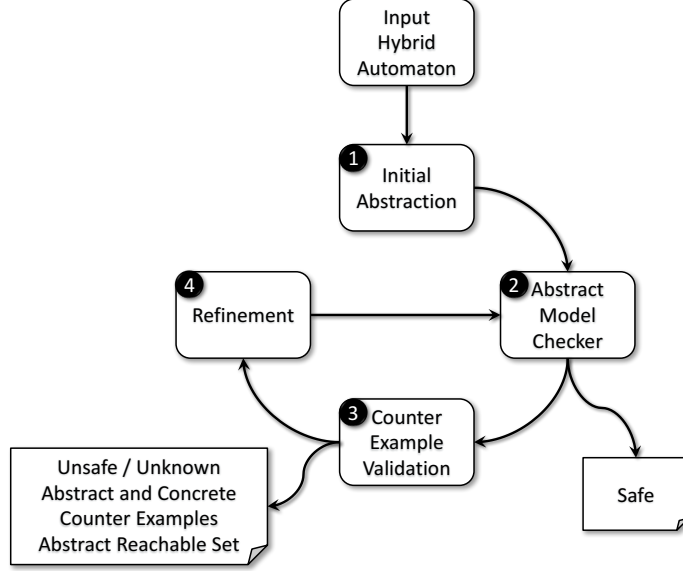


Figure 4.1: High Level CEGAR Loop

4.1 Computing Reachable Sets

Every CEGAR-based algorithm has four main parts [17]: 1. abstracting the concrete system, 2. model checking the abstract system, 3. validating the abstract counter-example, and 4. refining the abstract system. We explain parts of our algorithm regarding each of these parts in this section. Before that, Figure 4.1 shows these steps, and Algorithm 1 shows at a very high level what the steps of our algorithm are.

4.1.1 Time-Bounded Transitions

One step of every CEGAR algorithm is to validate a counter-example of an abstract system that is returned by the abstract model checker (Section 4.1.4). We do validation by running the counter-example of the abstract model checker against the concrete hybrid automaton. In our discussion, we will assume that for affine hybrid automata and non-linear polyhedral automata one can compute the continuous post of a set of states for an arbitrary amount of time. But this is not completely true. What we can do is to only compute approximations of the continuous post of a set of states. In addition, bounded error approximations can be computed only for a finite amount of time. Hence, we convert a hybrid automaton \mathcal{H} to another

Algorithm 1 High level steps of our CEGAR algorithm

Input: non-linear polyhedral automaton \mathcal{C} $\triangleright \mathcal{C}$ is concrete automaton. Def 2

Input: an unsafe set $U_{\mathcal{C}}$

Output: $\text{reach}(\mathcal{C}) \cap U_{\mathcal{C}} \stackrel{?}{=} \emptyset$ \triangleright See Safety Prob 4

1. $\mathcal{C} \leftarrow$ Add a self-loop to every location of \mathcal{C} \triangleright Sec 4.1.2
2. $P \leftarrow$ the initial partition of invariants in \mathcal{C} \triangleright Sec 4.1.2
3. $\mathcal{A} \leftarrow \alpha(\mathcal{C}, P)$ $\triangleright \mathcal{A}$ is abstract automaton. Def 16
4. $U_{\mathcal{A}} \leftarrow \alpha(U_{\mathcal{C}}, P)$ $\triangleright U_{\mathcal{A}}$ is called abstract unsafe set
5. $\tau = O^{\text{Poly}}(\mathcal{A}, U_{\mathcal{A}})$ $\triangleright O^{\text{Poly}}$ model checks polyhedral automata. Sec 4.1.3
6. $\triangleright \tau$ is an annotated counter-example. Sec 4.1.3
7. **while** $\tau \neq \emptyset$ **do** \triangleright while abstract system is unsafe
8. **if** τ is valid in \mathcal{C} **then return** ‘unsafe’ \triangleright Sec 4.1.4
9. $(q, p) \leftarrow$ abstract location that should be split \triangleright Sec 4.1.4
10. $p_1, p_2 \leftarrow$ sets that should be separated in (q, p) \triangleright Sec 4.1.4
11. refine $P(q)$ such that p_1 and p_2 gets separated \triangleright Sec 4.1.4
12. $\mathcal{A} \leftarrow \alpha(\mathcal{C}, P)$ \triangleright Sec 4.1.2
13. $U_{\mathcal{A}} \leftarrow \alpha(U_{\mathcal{C}}, P)$ \triangleright Sec 4.1.2
14. $\tau = O^{\text{Poly}}(\mathcal{A})$ \triangleright Sec 4.1.3
15. **end while**
16. **return** ‘safe’

hybrid automaton \mathcal{H}' with the same reachability information and with the additional property that in \mathcal{H}' , there is no time transition with a label larger than t , for some parameter $t \in \mathbb{R}_+$. With this transformation, we can compute bounded error approximations of the unbounded time post, since it is actually a continuous post over a bounded time t .

To construct \mathcal{H}' , we add a new variable z to \mathcal{H} . Variable z is a clock, initially 0, and on every edge, the guard imposes no constraints on z , while it is always reset to 0 after taking the edge. \mathcal{H}' also has a trivial self-loop on every location. Furthermore, invariant of \mathcal{H}' puts an upper bound on possible values of z which makes duration of any continuous trajectory bounded by t . Therefore, we know that unbounded time continuous post becomes equivalent to continuous post for at most t units of time. Finally, trivial self-loops guarantee that restricting continuous time transitions does not change reachable set of \mathcal{H} . Definition 14 shows how the new hybrid automaton is formally constructed.

Definition 14 (Bounding Time). For any non-linear polyhedral automaton \mathcal{H} and time bound $t: \mathbb{R}_+$, we construct a non-linear polyhedral automaton

\mathcal{H}' as follows:

- $X_{\mathcal{H}'} := X_{\mathcal{H}} \cup \{z\}$, assuming $z \notin X_{\mathcal{H}}$,
- $Q_{\mathcal{H}'} := Q_{\mathcal{H}}$,
- $\forall q \bullet X_{\mathcal{H}'}^{\text{init}}(q) := X_{\mathcal{H}}^{\text{init}}(q) \wedge (z = 0)$,
- $\forall q \bullet I_{\mathcal{H}'}(q) := I_{\mathcal{H}}(q) \wedge (z \leq t)$,
- $\forall q, x \bullet F_{\mathcal{H}'}(q, x) := F_{\mathcal{H}}(q, x)$ if $x \neq z$ and 1 otherwise, and
- $E_{\mathcal{H}'} := \{(s, d, l, r') \mid \exists r \bullet (s, d, l, r) \in E'' \wedge r' = r \wedge (z' = 0)\}$, where set E'' is the union of $E_{\mathcal{H}}$ and the set $\{(q, q, \emptyset, \bigwedge_{x : X_{\mathcal{H}}} x = x') \mid q : Q_{\mathcal{H}}\}$.

4.1.2 Abstraction

Input to our algorithm is a non-linear polyhedral automaton \mathcal{C} which we call the *concrete* hybrid automaton. The first step is to construct an *abstract* hybrid automaton \mathcal{A} which is a polyhedral automaton. The abstract hybrid automaton \mathcal{A} is obtained from the concrete hybrid automaton \mathcal{C} , by splitting the invariant of any location $q \in Q_{\mathcal{C}}$ into a finite number of cells of type $\mathbb{P}(X)$ and defining an abstract location for each of these cells which over-approximates the non-linear or affine dynamics in the cell by a polyhedral dynamics. Note, only flow of \mathcal{C} is abstracted in each of these cells to construct \mathcal{A} . This helps us simplify the abstraction. Definition 15 and Definition 16 formalize the way an abstract polyhedral automaton \mathcal{A} is constructed from \mathcal{C} .

Definition 15 (Invariant Partitions). For any hybrid automaton \mathcal{C} and function $P : Q \rightarrow 2^{\mathbb{P}(X)}$ we say P partitions invariants of \mathcal{C} iff the following conditions hold for any location $q : Q$:

- $\bigcup P(q) = I(q)$, meaning the union of cells in $P(q)$ covers the invariant of q .
- $\forall p_1, p_2 : P(q)$, p_1 and p_2 are either disjoint, or identical, or only share a common facet.

Definition 16 (Abstraction using Invariant Partitioning). For any non-linear polyhedral automaton \mathcal{C} and invariant partition $P : \mathbb{Q} \rightarrow 2^{\mathbb{P}(\mathbf{x})}$, $\alpha(\mathcal{C}, P)$ returns polyhedral automaton \mathcal{A} which is defined below:

- $\mathbb{Q}_{\mathcal{A}} = \{(q, p) \mid q \in \mathbb{Q}_{\mathcal{C}} \wedge p \in P(q)\},$
- $\mathbb{Q}_{\mathcal{A}}^{\text{init}} = \{(q, p) \in \mathbb{Q}_{\mathcal{A}} \mid q \in \mathbb{Q}_{\mathcal{C}}^{\text{init}}\},$
- $\mathbb{Q}_{\mathcal{A}}^{\text{bad}} = \{(q, p) \in \mathbb{Q}_{\mathcal{A}} \mid q \in \mathbb{Q}_{\mathcal{C}}^{\text{bad}}\},$
- $\mathbb{X}_{\mathcal{A}} = \mathbb{X}_{\mathcal{C}},$
- $\mathbb{I}_{\mathcal{A}}((q, p)) = p,$
- $\mathbb{E}_{\mathcal{A}} = \{((s, p_1), (d, p_2), g, j, r) \mid (s, d, g, j, r) \in \mathbb{E}_{\mathcal{C}} \wedge (s, p_1), (d, p_2) \in \mathbb{Q}_{\mathcal{A}}\},$
and
- $\mathbb{F}_{\mathcal{A}}((q, p), \nu) = \text{polyhull}(\bigcup_{\nu \in p} \mathbb{F}_{\mathcal{C}}(q, \nu)),$ where for any bounded set $S \subset \mathbf{X} \rightarrow \mathbb{R}$, $\text{polyhull}(S)$ is a polytope W such that $\forall \nu \in S. \nu \in W$ and for any sequence of bounded sets S_1, S_2, \dots , if the maximum distance of any two points in S_n converges to 0 then the maximum distance of any two points in the image of this sequence under polyhull converges to 0 as well.

In addition, we define function $\gamma_{\mathcal{A}}$ to map 1. every state in $\llbracket \mathcal{A} \rrbracket$ to a state in $\llbracket \mathcal{C} \rrbracket$, and 2. every edge in $\mathbb{E}_{\mathcal{A}}$ to an edge in $\mathbb{E}_{\mathcal{C}}$. Formally, for any $s = ((q, p), \nu) \in \mathbb{S}_{\llbracket \mathcal{A} \rrbracket}$ and $e = ((q_1, p_1), (q_2, p_2), r) \in \mathbb{E}_{\mathcal{A}}$, we define $\gamma_{\mathcal{A}}(s)$ to be (q, ν) and $\gamma_{\mathcal{A}}(e)$ to be (q_1, q_2, r) .

For each concrete location we will have one or more abstract locations. By making invariants of abstract locations small (and thus increasing the number of abstract locations) we want to be able to make behavior of \mathcal{A} as close as required to the behavior of \mathcal{C} . This requires trajectories to be always able to jump between two abstract locations when they correspond to a single concrete location. But we did not add any such edge to \mathcal{A} in Definition 16. Although defining the abstract system in this way just imposes an additional initial step to our algorithm, we find it very convenient not to introduce any edge in the abstract hybrid automata that corresponds to no edge in the concrete hybrid automata. Nonetheless, it is easy to see that if for every location $q \in \mathbb{Q}_{\mathcal{C}}$, $\mathbb{E}_{\mathcal{C}}$ contains a trivial edge (*i.e.* an edge with no guard and

no reset) from q to itself, abstracting \mathcal{C} using Definition 16 will produce a trivial edge between all abstract locations corresponding to a single concrete location. One can easily add these edges to \mathcal{C} in an initial step, so in the rest of this chapter, *Wlog.* we assume every location of \mathcal{C} has a trivial self loop. Finally, it is easy to see that these trivial self loops along with Definition 15 and Definition 16 introduce Zeno behavior in the abstract system (*i.e.* the abstract system can make an infinite number of discrete transitions in a finite amount of time), but our model checker can easily handle it. In fact since we check for a fixed-point, we believe our tool is not considerably affected by this type of behavior.

Proposition 17 (Over-Approximation). For any affine hybrid automaton \mathcal{C} and invariant partition P , $\mathcal{A} = \alpha(\mathcal{C}, P)$ is a polyhedral automaton which *over-approximates* \mathcal{C} , that is, $\text{reach}(\mathcal{C}) \subseteq \gamma_{\mathcal{A}}(\text{reach}(\mathcal{A}))$.

It is clear that if \mathcal{A} is safe then \mathcal{C} is also safe. Also, one can easily see that if P is defined as $P(q) = \{\mathcal{I}_{\mathcal{C}}(q)\}$ (for all $q \in \mathbb{Q}_{\mathcal{C}}$), it is a valid invariant partition of \mathcal{C} . It is actually what our algorithm always uses as the initial invariant partitioning, *i.e.*, initially we do not partition any invariant.

4.1.3 Counter-example and Model Checking Rectangular Automata

After an abstract hybrid automaton is constructed (initially and after any refinement), we have to model check it. In this section we define the notion of a counter-example and annotation of a counter-example, which we assume is returned by the abstract model checker O^{Poly} when it finds that the input hybrid automaton is unsafe.

Definition 18. For any hybrid automaton \mathcal{H} , a *counter-example* is a path e_1, \dots, e_n such that $\mathcal{S}e_1 \in \mathbb{Q}^{\text{init}}$ and $\mathcal{D}e_n \in \mathbb{Q}^{\text{bad}}$.

Definition 19. A counter-example π is called *valid* in \mathcal{H} iff \mathcal{H} has a run ρ and ρ has the same path as π . A counter-example that is not valid is called *spurious*.

Definition 20. An *annotation* for a counter-example $\pi = e_1, \dots, e_n$ of hybrid automaton \mathcal{H} is a sequence $\rho = S_0 \rightarrow S'_0 \xrightarrow{e_1} S_1 \rightarrow S'_1 \xrightarrow{e_2} \dots \xrightarrow{e_n} S_n \rightarrow S'_n$ such that the following conditions hold:

1. $\forall 0 \leq i \leq n \bullet \emptyset \neq S_i, S'_i \subseteq \mathbf{S}_{\llbracket \mathcal{H} \rrbracket},$
2. $\forall 0 \leq i \leq n \bullet S_i = \text{cpre}_{\mathcal{H}}(S'_i),$
3. $\forall 0 \leq i < n \bullet S'_i = \text{dpre}_{\mathcal{H}}^{e_{i+1}}(S_{i+1}),$
4. $S'_n = \mathbf{S}_{\llbracket \mathcal{H} \rrbracket}^{\text{bad}} \cap (\{\text{De}_n\} \times \mathbf{V}_{\mathcal{H}}).$

Condition 1 means that each S_i and S'_i in ρ are a non-empty set of states. Conditions 2 and 3 mean that sets of states in ρ are computed using backward reachability. Finally, condition 4 means that S'_n is the set of unsafe states in destination of e_n . Note that these conditions completely specify S_0, \dots, S_n and S'_0, \dots, S'_n from e_1, \dots, e_n and \mathcal{H} . Also, every S_i and S'_i is a subset of states corresponding to exactly one location.

In this chapter, we assume to have access to an oracle O^{Poly} that can correctly answer reachability problems when the hybrid automata are restricted to be polyhedral automata. If no unsafe location of \mathcal{A} is reachable from an initial location of it, $O^{\text{Poly}}(\mathcal{A})$ returns ‘safe’. Otherwise, it returns an annotated counter-example of \mathcal{A} .

4.1.4 Validating Abstract Counter-examples

For any invariant partition P and non-linear polyhedral automaton \mathcal{C} , if $O^{\text{Poly}}(\mathcal{A})$ (for $\mathcal{A} = \alpha(\mathcal{C}, P)$) returns ‘safe’, we know \mathcal{C} is safe. So the algorithm returns \mathcal{C} is ‘safe’ and terminates. On the other hand, if O^{Poly} finds \mathcal{A} to be unsafe it returns an annotated counter-example ρ of \mathcal{A} . Since \mathcal{A} is an over-approximation of \mathcal{C} , we cannot be certain at this point that \mathcal{C} is also unsafe. More precisely, if π is the path in ρ , we do not know whether $\gamma_{\mathcal{A}}(\pi)$ is a valid counter-example in \mathcal{C} or it is spurious. Therefore, we need to validate ρ in order to determine if it corresponds to any actual run from an initial location to an unsafe location in \mathcal{C} .

To validate ρ , an annotated counter-example of $\mathcal{A} = \alpha(\mathcal{C}, P)$, we run ρ on \mathcal{C} . More precisely, we create a sequence $\rho' = R_0 \rightarrow R'_0 \xrightarrow{e'_1} R_1 \rightarrow \dots \xrightarrow{e'_n} R_n \rightarrow R'_n$ where

1. $e'_i = \gamma_{\mathcal{A}}(e_i),$
2. $R_0 = \gamma_{\mathcal{A}}(S_0),$

3. $R'_i = \text{cpost}_C(R_i) \cap \gamma_{\mathcal{A}}(S'_i)$,
4. $R_i = \text{dpost}_C^{e'_i}(R'_{i-1}) \cap \gamma_{\mathcal{A}}(S_i)$.

Condition 1 states that edges in ρ' correspond to the edges in ρ as defined by the function $\gamma_{\mathcal{A}}$ in Definition 16. Condition 2 states that R_0 is just concrete states corresponding to S_0 . Note that R_0 is never empty. Condition 3 states that each R'_i is the intersection of two sets: 1. concrete states corresponding to abstract states in S'_i , and 2. continuous post of R_i . Condition 4 states that each R_i is the intersection of two sets: 1. concrete states corresponding to abstract states in S_i , and 2. discrete post of R'_{i-1} using e'_i . It is easy to see that for any i if R_i or R'_i becomes empty then for all $j > i$ both R_j and R'_j will be empty. Also, if R_i is empty then R'_i is empty too. Figure 4.2 depicts the situation when the counter-example is spurious and R'_i is the first empty set we reach during our validation. Proposition 21 proves that the first empty set (if any) is always R'_i for some i and not R_i .

Proposition 21. $R'_n = \emptyset$ in ρ' implies there exists i such that 1. $R'_i = \emptyset$, 2. $R_i \neq \emptyset$, 3. $\forall j < i. R_j, R'_j \neq \emptyset$, and 4. $\text{cpost}_C(R_i)$ and $\gamma_{\mathcal{A}}(S'_i)$ are nonempty disjoint sets.

Proof. We know that for all i neither S_i nor S'_i is empty. We prove that if $R'_{i-1} \neq \emptyset$ then $R_i \neq \emptyset$. By definition we have 1. $R'_{i-1} \subseteq \gamma_{\mathcal{A}}(S'_{i-1})$, and 2. $\forall \nu_1 \in S'_{i-1}. \exists \nu_2 \in S_i. \nu_1 \xrightarrow{e_i} \nu_2$. Note that we do not change guards and resets in abstraction. Therefore, R'_{i-1} is a subset of states that can take edge e'_i and after transition they go to $\gamma_{\mathcal{A}}(S_i)$, which means $R_i \neq \emptyset$. This proves the first three parts. Let i be such that those parts hold. Note that $\gamma_{\mathcal{A}}(S'_i) \neq \emptyset$, and if $R_i \neq \emptyset$ then $\text{cpost}_C(R_i) \neq \emptyset$. Furthermore, $\gamma_{\mathcal{A}}(S'_i)$ and $\text{cpost}_C(R_i)$ are disjoint, since, otherwise R'_i would be non-empty. \square

Lemma 22. The counter-example $\pi' = e'_1, \dots, e'_n$ of \mathcal{C} is valid iff $R'_n \neq \emptyset$.

Proposition 21 tells us that two sets $\text{cpost}_C(R_i)$ and $\gamma_{\mathcal{A}}(S'_i)$ are disjoint. Lemma 23 states a stronger result that there is a minimum distance $\epsilon > 0$ between those two sets, by exploiting the compactness of the two sets.

Lemma 23. There exists $\epsilon \in \mathbb{R}_+$ such that $\text{dist}_H(\text{cpost}_C(R_i), \gamma_{\mathcal{A}}(S'_i)) > \epsilon$.

Proof. Invariants, guards, and resets of the affine hybrid automaton \mathcal{C} are all closed and bounded (by definition). Hence, the invariants, guards, resets, and

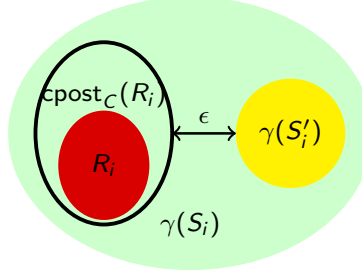


Figure 4.2: Validation and Refinement. S_i and S'_i are elements of annotated counter-example ρ . R_i and $\text{cpost}_C(R_i)$ are computed when ρ is validated. i is the smallest index for which $\text{cpost}_C(R_i)$ and $\gamma_{\mathcal{A}}(S'_i)$ are separated. Hence we need to refine \mathcal{A} in location i . Refinement should be done in such a way that for the result of refinement \mathcal{A}' we have $\text{cpost}_{\mathcal{A}'}(\gamma_{\mathcal{A}'}^{-1}(R_i)) \cap \gamma_{\mathcal{A}'}(S'_i) = \emptyset$. flow of the abstract rectangular automaton $\mathcal{A} = \alpha(C, P)$ are also closed and bounded. Therefore, each of the sets S_i, S'_i and R_i, R'_i are compact (closed and bounded). Since, $\text{cpost}_C(R_i)$ and $\gamma_{\mathcal{A}}(S'_i)$ are compact sets, there exists a minimum distance between them. \square

4.1.5 Refinement

Let us fix a concrete automaton \mathcal{C} , an invariant partition P , and an abstract automaton $\mathcal{A} = \alpha(\mathcal{C}, P)$. Suppose model checking \mathcal{A} reveals a counter-example π and its annotation ρ . If ρ is found to be spurious by the validation algorithm (in Section 4.1.4), then we need to refine the model \mathcal{A} by refining the invariant partition P . We will do this by refining the invariant of only a single location of \mathcal{A} . In this section we describe how to do this.

Since ρ is spurious, there is a smallest index i such that $R'_i = \emptyset$ (where the sets R_i, R'_i are as defined in Section 4.1.4); we will call this the *point of refinement* and denote it as $\text{por}_{\mathcal{C}, \mathcal{A}}(\rho)$. We will refine the location $(q, p) = \text{De}_i$ of \mathcal{A} by refining its invariant p . We know from Proposition 21, $\text{cpost}_C(R_i) \cap \gamma_{\mathcal{A}}(S'_i) = \emptyset$. However, the corresponding sets in the abstract system \mathcal{A} are not disjoint, that is, $\text{cpost}_{\mathcal{A}}(\gamma_{\mathcal{A}}^{-1}(R_i)) \cap S'_i \neq \emptyset$. Our refinement strategy is to find a partition for the location (q, p) such that in the refined model $R = \alpha(\mathcal{C}, P')$ (for some P'), S'_i is not reachable from R_i . In order to define the actual refinement, and to make this condition precise, we need to introduce some definitions.

Let \mathcal{C} , \mathcal{A} , R_i , S'_i , and (q, p) be as above. Let us denote by $\mathcal{C}_{q,p}$ the restriction of \mathcal{C} to the single location q with invariant p , i.e., $\mathcal{C}_{q,p}$ has only one location

q whose flow and invariant is the same as that of (q, p) in \mathcal{A} , and only transitions whose source and destination is q . We will say that an invariant partition P_r of $C_{q,p}$ *separates* R_i from S'_i iff in the automaton $\mathcal{A}_1 = \alpha(C_{q,p}, P_r)$, $\text{reach}_{\mathcal{A}_1}(\gamma_{\mathcal{A}_1}^{-1}(R_i)) \cap \gamma_{\mathcal{A}_1}^{-1}(\gamma_{\mathcal{A}}(S'_i)) = \emptyset$. In other words, the states corresponding to S'_i in \mathcal{A}_1 are not reachable from $\gamma_{\mathcal{A}_1}^{-1}(R_i)$ in \mathcal{A}_1 .

Refinement Strategy

Let P_r be an invariant partition of $C_{q,p}$ that separates R_i from S'_i . Define the invariant partition P' of \mathcal{C} as follows: $P'(q') = P(q')$ if $q' \neq q$, and $P'(q) = (P(q) \setminus \{p\}) \cup P_r(q)$. The new abstract automaton will be $R = \alpha(\mathcal{C}, P')$. Observe that R is a refinement of \mathcal{A} (since the invariant partition is refined), and the relationship between the locations and edges of the two automata is characterized by a function $\alpha_{R,\mathcal{A}}(\cdot)$ defined as follows. For a location (q', p') , $\alpha_{R,\mathcal{A}}(q', p') = (q', p')$ if either $q' \neq q$, or $p' \not\subseteq p$, and $\alpha_{R,\mathcal{A}}(q', p') = (q, p)$ otherwise. Having defined the mapping between locations, the mapping between edges is its natural extension:

$$\alpha_{R,\mathcal{A}}((q_1, p_1), (q_2, p_2), g, j, r) = (\alpha_{R,\mathcal{A}}(q_1, p_1), \alpha_{R,\mathcal{A}}(q_2, p_2), g, j, r)$$

The goal of the refinement strategy outlined above is to ensure that a given counter-example π is eventually eliminated, if the abstract model checker generates it sufficiently many times. To make this statement precise and to articulate the nature of progress we need to first identify when a counter-example of R corresponds to a counter-example of \mathcal{A} . Observe that a path π of \mathcal{A} can “correspond” to a longer path π' in R , where previous sojourn in location (q, p) in π , now corresponds to a path in π' that traverses the newly created locations by partitioning p . Recall that we are assuming that $\text{por}_{C,\mathcal{A}}(\rho) = i$, where ρ is the annotation corresponding to π . We will say that a counter-example $\pi' = e'_1, e'_2, \dots, e'_m$ *corresponds* to counter-example $\pi = e_1, e_2, \dots, e_n$, if there exists k , $0 \leq k \leq m - i$, such that 1. for all $j \leq i$, $\alpha_{R,\mathcal{A}}(e'_j) = e_j$, 2. for all $j > i + k$, $\alpha_{R,\mathcal{A}}(e'_j) = e_{j-k}$, and 3. for all $i < j \leq i + k$, source and destination of $\alpha_{R,\mathcal{A}}(e'_j)$ is (q, p) . If π' corresponds to π , we will call k its witness. Using this notion of correspondence, we are ready to state what our refinement achieves.

Proposition 24. Let π be a counter-example of \mathcal{A} and ρ its annotation.

Let R be the refinement constructed by our strategy after ρ is found to be spurious. Let π' be a counter-example of R that corresponds to π , and let ρ' be its annotation. Then, $\text{por}_{C,R}(\rho') < \text{por}_{C,\mathcal{A}}(\rho)$.

Proof. Let $i = \text{por}_{C,\mathcal{A}}(\rho)$, and let R_i and S'_i be sets as defined in the validation algorithm. Let k be the witness for the correspondence between π' and π . Observe that our refinement strategy ensures that the set of states that can reach S'_i through the path $\pi'[i, i+k]$ is disjoint from $\gamma_{\mathcal{A}}^{-1}(R_i)$. Hence if the sequence $U_0, U'_0, U_1, \dots, U_m$ is computed by the validation algorithm for ρ' , we know that $U_i = \emptyset$. Using Proposition 21, we can conclude that $\text{por}_{C,R}(\rho') < i$. \square

The above proposition implies that a counter-example π can appear only finitely many times in the CEGAR loop. This is because the point of refinement of any π' in R corresponding to π in \mathcal{A} is strictly smaller.

Next, we claim that a partition satisfying the refinement strategy always exists. It relies on the following observation from [72] which states that the reach set of a non-linear dynamical system can be approximated to within any ϵ by a rectangular hybridization over a bounded time interval.

Theorem 25 ([72]). Let \mathcal{H} be a non-linear hybrid automaton with a single location such that there is a bound T on the time for which the system can evolve in the location. Then, for any $\epsilon > 0$, there exists an invariant partition P of \mathcal{H} such that $\text{dist}_{\mathbb{H}}(\text{reach}(\mathcal{H}), \text{reach}(\alpha(\mathcal{H}, P))) < \epsilon$.

Corollary 26 (Existence of Refinement). There always exists a partition P' that separates R_i and S'_i .

Proof. The result follows from Theorem 25 and Lemma 23. \square

4.1.6 Validation Approximation

In order to validate a counter-example, we assumed that the continuous post for bounded time of a set of states in an non-linear hybrid automaton can be computed exactly.

we assumed to be able to exactly compute continuous post of a set of states in the affine hybrid automaton for a finite amount of time. But the best one can actually hope for is computing over and under approximation of this

set. In this section we show that being able to approximate the continuous post is enough for our algorithm. For any hybrid automaton \mathcal{H} , set of states $S \subseteq \mathbb{S}_{\llbracket \mathcal{H} \rrbracket}$, edge $e \in \mathbb{E}_{\mathcal{H}}$, and parameter $\epsilon \in \mathbb{R}_+$ we define the following functions:

- $\text{cpost}_{\text{over}}^\epsilon(S)$ is an over-approximation of $\text{cpost}(S)$. If $\text{cpost}_{\text{over}}^\epsilon(S)$ returns S' then we know $\text{cpost}(S) \subseteq S'$ and $\text{dist}_{\mathcal{H}}(S', \text{cpost}(S)) < \epsilon$.
- $\text{cpost}_{\text{under}}^\epsilon(S)$ is an under-approximation of $\text{cpost}(S)$. If $\text{cpost}_{\text{under}}^\epsilon(S)$ returns S' then we know $\text{cpost}(S) \supseteq S'$ and $\text{dist}_{\mathcal{H}}(S', \text{cpost}(S)) < \epsilon$.

During the validation procedure, instead of computing ρ' we compute ρ_o and ρ_u . They are computed exactly as ρ' , except that in ρ_o and ρ_u , instead of cpost , we respectively use $\text{cpost}_{\text{over}}^\epsilon$ and $\text{cpost}_{\text{under}}^\epsilon$. Let us denote the last elements of ρ_o and ρ_u respectively by R'_n and U'_n . If U'_n is non-empty, we know ρ represents at least one valid counter-example. Therefore, the algorithm outputs ‘unsafe’ and terminates. If U'_n is empty but R'_n is non-empty, it means ϵ is too big. Therefore, the algorithm repeats itself using $\frac{\epsilon}{2}$. If R'_n is empty, it means all counter-examples in ρ are spurious. Therefore, too much over-approximation is deployed in \mathcal{A} and the abstraction \mathcal{A} is refined as outlined in Section 4.1.5.

Lemma 27. Given a counter-example π of \mathcal{A} , if $\gamma_{\mathcal{A}}(\pi)$ is spurious, then there exists an $\epsilon > 0$ for which R'_n is empty.

The above lemma states that if the abstract counter-example is spurious, then the same will be detected by our algorithm. This is a direct consequence of Lemma 23. In Section 4.2, we show that arbitrarily over-approximation of the reachable set is possible for non-linear polyhedral automata.

4.1.7 Need for Forward and Backward Reachability

In a CEGAR framework, validating a counter-example typically involves just performing a forward search (with respect to the transitions in the counter-example) in the concrete system; henceforth called the *standard validation algorithm*. However, in the validation algorithm outlined in Section 4.1.4 is different. The forward search in the concrete system uses a sequence of sets of abstract states computed using backward reachability in abstract model

(i.e., the annotated counter-example). Thus, validation for us involves doing a backward search in the abstract model, *and* a forward search in the concrete model. The reason for doing this is because the standard validation algorithm fails to identify the correct refinement step in our case, where the counter-example corresponds to infinitely many executions.

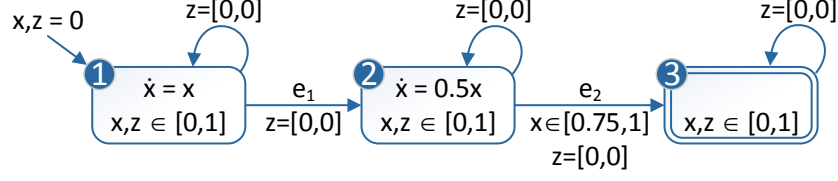
We illustrate this through an example affine hybrid automaton \mathcal{C} shown in Figure 4.3a. The automaton has 3 states $\{1, 2, 3\}$, with 1 as the only initial location, and 3 as the only unsafe location. The automaton has two variables x and z . z is a clock that is used to ensure that a discrete transition is taken every 1 unit of time; invariants of z in each location, and its guards and resets on every transition are set to ensure this. The dynamics of variable x in 1 are given by the equation $\dot{x} = x$, while that in location 2 are given by $\dot{x} = 0.5x$; the dynamics in 3 is not important as that is the unsafe location. The invariant in all locations is $x \in [0, 1]$ and $z \in [0, 1]$. There are 5 transitions: self loop on every location, transition e_1 from 1 to 2 and the transition e_2 from 2 to 3. All transitions reset z to 0, and leave x unchanged. All transitions except transition e_2 are always enabled; the transition e_2 is enabled when $x \in [0.75, 1]$. Assuming that the initial value of x is 0 in location 1³, the automaton is safe. This is because no matter how many discrete transitions are taken, the value of x remains 0 in both locations 1 and 2, and so the transition from 2 to 3 is never taken.

Consider the trivial invariant partition P for this automaton that leaves the invariant for each location intact. The rectangular automaton $\mathcal{A} = \alpha(\mathcal{C}, P)$ is shown in Figure 4.3b. The only difference between \mathcal{A} and \mathcal{C} is in the dynamics for variables x in locations 1 and 2 — the dynamics in 1 is given by $\dot{x} \in [0, 1]$ and in 2 by $\dot{x} \in [0, \frac{1}{2}]$. Observe that \mathcal{A} is unsafe because of the execution

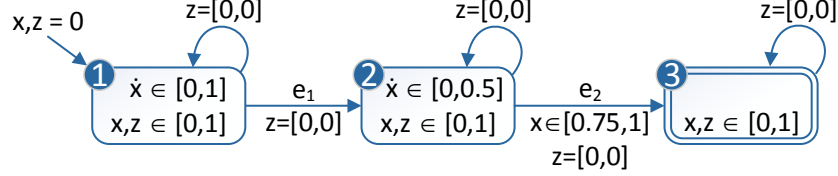
$$(1, x = 0) \xrightarrow{0.5} (1, x = 0.5) \xrightarrow{e_1} (2, x = 0.5) \xrightarrow{0.5} (2, x = 0.75) \xrightarrow{e_2} (3, x = 0.75)$$

In the above execution, we have skipped the value of variable z , since it does not play an important role. Thus, the abstract counter-example is $\pi = e_1, e_2$.

³Our model does not have initial values for continuous variables. But having initial values can easily be ensured by adding a new initial location, where the invariant for the variables is constrained to be the initial value. We did not do this here to keep the number of locations small.



(a) Affine hybrid automaton \mathcal{C}



(b) Rectangular automaton $\mathcal{A} = \alpha(\mathcal{C}, P)$

Figure 4.3: Need for annotated counter-examples in refinement. $\dot{z} = 1$ in all locations.

The annotated counter-example ρ corresponding to π is

$$\begin{aligned} (1, x \in [0, 0]) &\xrightarrow{\leq^1} (1, x \in [0.25, 1]) \xrightarrow{e_1} (2, x \in [0.25, 1]) \\ &\xrightarrow{\leq^1} (2, x \in [0.75, 1]) \xrightarrow{e_2} (3, x \in [0.75, 1]) \end{aligned}$$

Now doing the standard validation algorithm for the counter-example π using forward search in \mathcal{C} results in the following sequence.

$$(1, x \in [0, 0]) \xrightarrow{\leq^1} (1, x \in [0, 0]) \xrightarrow{e_1} (2, x \in [0, 0]) \xrightarrow{\leq^1} (2, x \in [0, 0]) \xrightarrow{e_2} (3, x \in \emptyset)$$

The counter-example π is spurious because the set $(2, x \in [0, 0])$ does not intersect the guard of e_2 suggesting that the dynamics in location 2 needs to be refined. How do we refine? We need to ensure that in the new abstract model (after refinement), there is no way to reach the guard of e_2 from the state $(2, x = 0)$, which is the entry state for location 2. But notice that the set of states reachable in \mathcal{A} from $(2, x = 0)$ is $(2, x \in [0, 0.5])$ which is already disjoint from guard of e_2 . So no matter how we refine \mathcal{A} will not make any progress from the standpoint of eliminating the counter-example π .

On the other hand, our validation algorithm will correctly identify that what needs to be refined is the dynamics in location 1 (and not 2). Recall that our validation algorithm does a forward search in \mathcal{C} , and each post computation is intersected with the corresponding set in the annotated counter-

example ρ . This will result in the following sequence.

$$(1, x \in [0, 0]) \xrightarrow{\leq 1} (1, x \in \emptyset) \xrightarrow{e_1} (2, x \in \emptyset) \xrightarrow{\leq 1} (2, x \in \emptyset) \xrightarrow{e_2} (3, x \in \emptyset)$$

Thus, the reason π is spurious is because the set of states reachable in \mathcal{C} from $(1, x = 0)$ (without any discrete transitions) is $(1, x = 0)$ which is disjoint from the set of all abstract states $(1, x \in [0.25, 1])$ that can exhibit the transitions e_1, e_2 . Thus, we need to refine the dynamics in location 1 in such a way that the set of states reachable from $(1, x = 0)$ (initial state) is disjoint from the set $(1, x \in [0.25, 1])$. This is easily achieved by splitting the invariant of location 1 into $[0, 0.2]$ and $[0.2, 1]$.

4.2 Arbitrary Approximation of Reachable Set for Non-linear Dynamics

In Section 4.1 we took an annotated abstract counter-example $\tau = S_0 \rightarrow S'_0 \xrightarrow{e_1} S_1 \rightarrow S'_1 \xrightarrow{e_2} \dots \xrightarrow{e_n} S_n \rightarrow S'_n$ and constructed $\tau' = R_0 \rightarrow R'_0 \xrightarrow{e'_1} R_1 \rightarrow R'_1 \xrightarrow{e'_2} \dots \xrightarrow{e'_n} R_n \rightarrow R'_n$. There are no known algorithms for constructing R_i and R'_i for non-linear dynamics. In [65] we proved that being able to arbitrarily over-approximate the sets R_i and R'_i is enough to detect spurious counter-examples.

Proposition 28. For any $i \in \{0, \dots, n\}$, sets S_i , S'_i , R_i , and R'_i are all compact.

Proof. S_i and S'_i are both subsets of the invariant of some location in \mathcal{A} and hence they are bounded. Also, initial and unsafe states, invariants, and transition relations are all closed sets. Furthermore, since discrete and continuous abstract posts of a polyhedron can be computed using Fourier-Motzkin variable elimination method, and when all input inequalities are of the same type, this method does not produce any inequality of the opposite type, we conclude S_i and S'_i are both closed sets as well.

Note that $R_0 = \gamma_{\mathcal{A}}(S_0)$ and $\gamma_{\mathcal{A}}$ of a compact set is compact. Therefore, R_0 is compact. Also the intersection of any two compact sets is compact. For any two sets $A, B \subseteq \mathbb{R}^{\mathbf{x}}$, concatenation of A with B is denoted by $A \frown B$ and is subset of $\mathbb{R}^{\mathbf{x} \cup \mathbf{x}'}$. A point ν is in $A \frown B$ iff there are points $\nu_A : A$ and $\nu_B : B$ such that $\forall x : \mathbf{x} \bullet \nu(x) = \nu_A(x) \wedge \nu(x') = \nu_B(x')$. It is easy to see that

the concatenation of any two compact sets is also compact. Furthermore, for any set of variable Y and set of points $A \subseteq \mathbb{R}^x$, if A is compact then $\exists Y \bullet A$ is also compact.

For concrete discrete post, we know $R'_{i-1} \sim \mathbf{I}(\text{De}'_i)$ and Re'_i are compact. Therefore, $R_i = (\exists \mathbf{X} \bullet (R'_{i-1} \sim \mathbf{I}(\text{De}'_i)) \cap \text{Re}'_i) \cap \gamma_{\mathcal{A}}(S_i)$ is compact. For concrete continuous post, we know continuous image of a compact set is compact. We know that the flows are continuous, which means that the trajectories are also continuous functions. Therefore, reach tube set of R_i , which is defined as $\{(\nu, t) : (\mathbb{R}^x, \mathbb{R}) \mid \exists \nu' : R_i \bullet \nu' \xrightarrow{t} \nu\}$, is compact. Since existential quantification of a compact set is compact, the reachable set is compact as well. \square

dReach [40] is a tool that takes as input a non-linear hybrid automaton \mathcal{H} , an unsafe set of states U , a bound N on number of discrete transitions, and a time-bound T . The tool checks whether U can be reached in \mathcal{H} using at most N number of discrete transitions and within time bound T . To do this, **dReach** reduces this problem into the satisfiability of a first order logic (FOL) formula with atoms of the form $f(\mathbf{X}) \leq 0$, where the left-hand-side is either 1. finite addition or subtraction of functions used in \mathcal{H} to specify initial states, invariants, and transition relations, 2. finite addition or subtraction of functions used in the specification of U , or 3. it is of the form $x_1 - x_2 - \int_0^t g(\mathbf{X}) d\mathbf{X}$ or its negation, where $g(\mathbf{X})$ is a flow function⁴. Since all inequalities in formulas defining R_i and R'_i are non-strict, there will be no atom of the form $f(\mathbf{X}) < 0$. Quantification is of the form $\exists \mathbf{X}_1, \dots, \mathbf{X}_n, t_1, \dots, t_n \bullet \forall t'_1, \dots, t'_n \bullet \phi$, where ϕ is a quantifier free formula. The answer to the question “whether U can be reached in \mathcal{H} ” is *yes* iff the formula constructed by **dReach** is satisfiable. This is where **dReach** uses another tool called **dReal** [73], which takes a first order formula ϕ and parameter $\delta : \mathbb{R}_+$ as inputs and returns either **unsat** or δ -**sat**. If **dReal** returns **unsat**, we know the exact formula is unsatisfiable and hence U is not reachable in \mathcal{H} after at most T units of times and at most N number of discrete transitions. However, if **dReal** returns δ -**sat**, we only know if every atomic formula $f(x) \leq 0$ is replaced by $f(x) \leq \delta$, the result formula will be satisfiable. Recall that $\delta : \mathbb{R}_+$ is a parameter to **dReal** that can be made arbitrary small. Finally, **dReal** assumes all variables are bounded and **dReach** provides all these bounds as input to the tool.

⁴It is also possible that the tool applies some simplifications, but it has no consequence in our discussion and we ignore it to make the discussion simpler.

We convert counter-example validation problem into reachability problem of a hybrid automaton such that the counter-example is valid iff U is reachable in \mathcal{H} after at most T units of times and at most N number of discrete transitions.

For any formula ϕ and parameter $\delta: \mathbb{R}$, we use ϕ^δ to refer to the formula in which every atomic formula $f(x) \leq 0$ is replaced by $f(x) \leq \delta$. Also, we use $\llbracket \phi \rrbracket$ to refer to the set of points that satisfy ϕ . Note that for any $\delta: \mathbb{R}_{\geq 0}$ we have $\llbracket \phi \rrbracket \subseteq \llbracket \phi^\delta \rrbracket$. For any set S and parameter $\epsilon: \mathbb{R}_+$, we say S' ϵ -over-approximates S iff $S \subseteq S' \subseteq B_\infty^\epsilon(S)$. For any formula ϕ , we say $\llbracket \phi \rrbracket$ can be arbitrarily over-approximated iff for any $\epsilon: \mathbb{R}_+$ there is $\delta: \mathbb{R}_+$ such that $\llbracket \phi^\delta \rrbracket$ ϵ -over-approximates $\llbracket \phi \rrbracket$. Unfortunately, **dReal** and hence **dReach** do not directly support arbitrarily over-approximation of a reachable set. Meaning, none of these tools tries to find a $\delta: \mathbb{R}_+$ such that every point in $\llbracket \phi^\delta \rrbracket$ becomes ϵ -close to a point in $\llbracket \phi \rrbracket$. However, in this section, we prove arbitrarily over-approximation is possible in our case.

Lemma 29. For any two compact sets $\llbracket A \rrbracket$ and $\llbracket B \rrbracket$, if $\llbracket A \rrbracket$ and $\llbracket B \rrbracket$ can both be arbitrary approximated then $\llbracket A \rrbracket \cap \llbracket B \rrbracket = \llbracket A \wedge B \rrbracket$ can be arbitrary approximated as well.

Proof. For the purpose of contradiction, suppose $\exists \epsilon: \mathbb{R}_+ \cdot \forall \delta: \mathbb{R}_+ \cdot \llbracket A^\delta \wedge B^\delta \rrbracket \not\subseteq B_\infty^\epsilon(\llbracket A \wedge B \rrbracket)$. For any $n: \mathbb{N}$, let $\epsilon_n := \frac{\epsilon}{2^n}$ and let $\delta_n: \mathbb{R}_+$ be such that $\llbracket A^{\delta_n} \rrbracket \subseteq B_\infty^{\epsilon_n}(\llbracket A \rrbracket)$ and $\llbracket B^{\delta_n} \rrbracket \subseteq B_\infty^{\epsilon_n}(\llbracket B \rrbracket)$. Also, let x_n be an arbitrary element of the non-empty set $\llbracket A^{\delta_n} \wedge B^{\delta_n} \rrbracket \setminus B_\infty^\epsilon(\llbracket A \wedge B \rrbracket)$. Since $\llbracket A \rrbracket$ is bounded, there is a sequence $m_0 < m_1 < m_2 < \dots$ such that $x_{m_1}, x_{m_2}, x_{m_3}, \dots$ converges to x^* .

Since for any $n: \mathbb{N}$, $\llbracket A^{\delta_n} \wedge B^{\delta_n} \rrbracket = \llbracket A^{\delta_n} \rrbracket \cap \llbracket B^{\delta_n} \rrbracket$, and since $\llbracket A \rrbracket$ and $\llbracket B \rrbracket$ are both closed sets, $x^* \in \llbracket A \rrbracket \cap \llbracket B \rrbracket = \llbracket A \wedge B \rrbracket$. There are infinitely many n such that $d_\infty(x_{m_n}, x^*) \leq \epsilon$ and any one of them is a contradiction to the fact that $x_{m_n} \notin B_\infty^\epsilon(\llbracket A \wedge B \rrbracket)$. \square

Lemma 30. For any non-empty polytope P over an arbitrary set of variables X , $\llbracket P \rrbracket$ can be arbitrarily over-approximated.

Proof. Obviously, $\forall \delta: \mathbb{R}_{\geq 0} \cdot \llbracket P \rrbracket \subseteq \llbracket P^\delta \rrbracket$. Using contradiction, we first prove

$$\forall \epsilon, \delta: \mathbb{R}_+ \cdot \left(\llbracket P^\delta \rrbracket \setminus B_\infty^\epsilon(P) \neq \emptyset \right) \Rightarrow \left(\left(\llbracket P^\delta \rrbracket \setminus B_\infty^\epsilon(P) \right) \cap B_\infty^{2\epsilon}(P) \neq \emptyset \right)$$

Suppose $\exists x: \llbracket P^\delta \rrbracket \cdot x \notin B_\infty^\epsilon(P) \wedge x \notin B_\infty^{2\epsilon}(P)$. We know $\exists x': \llbracket P \rrbracket \subseteq \llbracket P^\delta \rrbracket$. Since $\llbracket P^\delta \rrbracket$ is a convex set, $\forall \lambda: [0, 1] \cdot \lambda x + (1-\lambda)x' = x' + \lambda(x-x') \in \llbracket P^\delta \rrbracket$. We

know $d_\infty(x', x' + \lambda(x - x')) = \lambda d_\infty(x, x') > \lambda 2\epsilon$. Let $\lambda := \frac{1.5\epsilon}{d_\infty(x, x')}$. Therefore, $x'' := x' + \lambda(x - x') \in \llbracket P^\delta \rrbracket$ and has distance at most 1.5ϵ to $\llbracket P \rrbracket$ which is a contradiction.

We finish the prove of this case by another proof by contradiction. Suppose $\exists \epsilon : \mathbb{R}_+ \bullet \forall \delta : \mathbb{R}_+ \bullet \exists x : \llbracket P^\delta \rrbracket \bullet x \notin B_\infty^\epsilon(\llbracket P \rrbracket)$. Fix ϵ and for any $n : \mathbb{N}_+$ let δ_n be $\frac{1}{n}$ and $x'_n : \llbracket P^{\delta_n} \rrbracket \setminus B_\infty^\epsilon(\llbracket P \rrbracket)$ be an arbitrary element. Using what we just proved, let $x_n : (\llbracket P^{\delta_n} \rrbracket \setminus B_\infty^\epsilon(\llbracket P \rrbracket)) \cap B_\infty^{2\epsilon}(\llbracket P \rrbracket)$ be an arbitrary element as well. Since $B_\infty^{2\epsilon}(\llbracket P \rrbracket)$ is bounded, there is a sequence $m_1 < m_2 < \dots$ such that x_{m_1}, x_{m_2}, \dots converges to x^* . We know $x^* \in \llbracket P \rrbracket$. Also, there are infinitely many indices n such that $d_\infty(x_{m_n}, x^*) \leq \epsilon$ and any one of them contradicts the fact $x_{m_n} \notin B_\infty^\epsilon(\llbracket P \rrbracket)$. \square

Proposition 31. For any FOL formula ϕ and set of variables Y , if $\llbracket \phi \rrbracket$ can be arbitrarily over-approximated then $\llbracket \exists Y \bullet \phi \rrbracket$ can be arbitrarily over-approximated as well.

Recall that for any two sets $A, B \subseteq \mathbb{R}^{\mathbf{X}}$, concatenation of A with B is denoted by $A \smallfrown B$ and is subset of $\mathbb{R}^{\mathbf{X} \cup \mathbf{X}'}$. A point ν is in $A \smallfrown B$ iff there are points $\nu_A : A$ and $\nu_B : B$ such that $\forall x : \mathbf{X} \bullet \nu(x) = \nu_A(x) \wedge \nu(x') = \nu_B(x')$.

Proposition 32. For any two FOL formulas ϕ and ψ , if both $\llbracket \phi \rrbracket$ and $\llbracket \psi \rrbracket$ can be arbitrarily over-approximated then $\llbracket \phi \rrbracket \smallfrown \llbracket \psi \rrbracket$ can be arbitrarily over-approximated as well.

Theorem 33. For any index $i : \{0, \dots, n\}$, sets $\llbracket R_i \rrbracket$ and $\llbracket R'_i \rrbracket$ can be arbitrarily over-approximated.

Proof. Inclusions $\llbracket R_i \rrbracket \subseteq \llbracket R_i^\delta \rrbracket$ and $\llbracket R'_i \rrbracket \subseteq \llbracket R_i'^\delta \rrbracket$ are trivially true for any $\delta : \mathbb{R}_{\geq 0}$. Therefore, we only focus on the other inclusions. Proof is by induction on i .

1. Base of Induction: $R_0 = \gamma_{\mathcal{A}}(S_0)$ and $\gamma_{\mathcal{A}}(S_0)$ is a polytope. Lemma 30 completes the proof. Next we consider different cases for the inductive step.
2. Intersection: This case has already been proved in Lemma 29.
3. Discrete Post: Using Lemma 30 we know $\mathbf{I}(\text{De}'_i)$ can be arbitrarily over-approximated. Using the inductive hypothesis, we know R'_{i-1} can be arbitrarily over-approximated. Using Proposition 32 we know the

concatenation of these two sets can be arbitrarily over-approximated. Using Lemma 30 we know $R(e'_i)$ can be arbitrarily over-approximated. Using the previous case, we know R_i which is intersection of the last two sets can be arbitrarily over-approximated as well.

4. Continuous Post: Suppose we are in location q with invariant I , flow $\dot{f}: V \rightarrow V$, and closed set $\llbracket R \rrbracket$ which is either the initial set or it is reached after the last discrete transition. Using induction hypothesis, we assume $\llbracket R \rrbracket$ can be arbitrary over-approximated. We want to prove the set of points that can be reached from $\llbracket R \rrbracket$ using trajectories defined by \dot{f} can be arbitrary over-approximated as well. Suppose durations of continuous transitions cannot be larger than $T: \mathbb{R}_+$. Since $\llbracket I \rrbracket$ is compact and \dot{f} is a continuous function, \dot{f} is bounded as well⁵. This means $f(x_0, t) := x_0 + \int_0^t \dot{f}(x) dt$, *i.e.* trajectories, are λ -Lipchitz continuous for some $\lambda: \mathbb{R}_+$ (*wlog.*, assume $\lambda \geq 1$).

We first prove

$$\begin{aligned} \forall \epsilon: \mathbb{R}_+ \bullet \forall t: [0, T] \bullet \exists \epsilon': (0, \epsilon) \bullet \forall x: B_\infty^{\epsilon'}(\llbracket R \rrbracket) \setminus \llbracket R \rrbracket \bullet \\ (\forall t': [0, t] \bullet f(x, t') \in \llbracket I \rrbracket) \Rightarrow \\ \exists x': \llbracket R \rrbracket \cap B_\infty^\epsilon(x) \bullet \forall t': [0, t] \bullet f(x', t') \in \llbracket I \rrbracket \end{aligned}$$

For the purpose of contradiction, suppose

$$\begin{aligned} \exists \epsilon: \mathbb{R}_+ \bullet \exists t: [0, T] \bullet \forall \epsilon': (0, \epsilon) \bullet \exists x: B_\infty^{\epsilon'}(\llbracket R \rrbracket) \setminus \llbracket R \rrbracket \bullet \\ (\forall t': [0, t] \bullet f(x, t') \in \llbracket I \rrbracket) \wedge \\ \forall x': \llbracket R \rrbracket \cap B_\infty^\epsilon(x) \bullet \exists t': [0, t] \bullet f(x', t') \notin \llbracket I \rrbracket \end{aligned}$$

Fix ϵ and t . For any $n: \mathbb{N}_+$ define $\epsilon'_n := \frac{\epsilon}{2n}$ and let $x_n: B_\infty^{\epsilon'_n}(\llbracket R \rrbracket) \setminus \llbracket R \rrbracket$ be a witness for the previous formula. Since $B_\infty^\epsilon(\llbracket R \rrbracket)$ is bounded and $x_n \in B_\infty^\epsilon(\llbracket R \rrbracket)$, there is a sequence $m_1 < m_2 < m_3 < \dots$ such that $x_{m_1}, x_{m_2}, x_{m_3}, \dots$ converges to x^* . We know $x^* \in \llbracket R \rrbracket$ and since $\llbracket I \rrbracket$ is closed and f is Lipchitz continuous, we also know $\forall t': [0, t] \bullet f(x^*, t') \in \llbracket I \rrbracket$. There are infinitely many n such that $d_\infty(x^*, x_{m_n}) \leq \epsilon$ and any one of them can witness the contradictory fact $\exists n: \mathbb{N}_+ \bullet x^* \in B_\infty^\epsilon(x_{m_n})$.

Suppose we want to ϵ -over-approximate the reachable set after con-

⁵We need \dot{f} to be continuous not only in $\llbracket I \rrbracket$, but also in an arbitrary small neighborhood of it.

tinuous transition. Let $\epsilon' := \frac{\epsilon}{4\lambda}$ and divide the set $B_\infty^{\epsilon'}(\llbracket R \rrbracket)$ into finite number of closed sets such that all points in a set are ϵ' -close to each other. We call each one of these sets a box. Finally, divide $[0, T]$ at times $0 = t_0 < t_1 < t_2 < \dots < t_m = T$ such that $\forall i < m \bullet t_{i+1} - t_i \leq \epsilon'$. We construct a finite sequence $\epsilon' = \epsilon''_1 > \epsilon''_2 > \dots > \epsilon''_r > 0$ such that $\llbracket \text{cpost}_{\mathcal{C}_{q, \mathbf{I}}}^{\epsilon''_{r+1}}(B_\infty^{\epsilon''_r}(R)) \rrbracket \subseteq B_\infty^\epsilon(\llbracket \text{cpost}_{\mathcal{C}_{q, \mathbf{I}}}(R) \rrbracket)$, where ϵ''_{r+1} is what Lemma 30 returns for the inputs \mathbf{I} and ϵ''_r .

For any box B , since $\forall x : B \bullet \mathbf{d}_\infty(x, \llbracket R \rrbracket) \leq \epsilon' \leq \frac{\epsilon}{4}$ and $\llbracket R \rrbracket$ is a subset of exact reachable set after continue transition, at time $t_0 = 0$, all points in B and therefore the whole $B_\infty^{\epsilon''}(\llbracket R \rrbracket)$ can be recognized as reachable states in the ϵ -over-approximation. Next for any box B , we inductively consider t_i for $0 < i \leq m$. There are three cases:

- $\exists \eta : \mathbb{R}_+ \bullet \forall x : B \bullet \exists t : [0, t_i] \bullet \mathbf{d}_\infty(f(x, t), \llbracket \mathbf{I} \rrbracket) > \eta$. Using Lemma 30, we find a smaller value for the next ϵ''_j such that any trajectory in B that leaves $B_\infty^\eta(\llbracket \mathbf{I} \rrbracket)$ will be rejected (*i.e.* all trajectories in B). This means no trajectory in B can have duration t_i or larger. Note that during $[t_{i-1}, t_i]$ each trajectory can move at most $\frac{\epsilon}{4}$. By inductive hypothesis, we know all points reachable from B at time t_{i-1} are $\frac{\epsilon}{2}$ -close to the exact reachable set. Therefore, all points reachable from B during $[t_{i-1}, t_i]$ are ϵ -close to the exact reachable set.
- $\exists x : B \cap \llbracket R \rrbracket \bullet \forall t : [0, t_i] \bullet f(x, t) \in \llbracket \mathbf{I} \rrbracket$. Since $\forall x' : B \bullet \mathbf{d}_\infty(x, x') \leq \epsilon'$, we conclude $\forall x' : B, t : [0, t_i] \bullet \mathbf{d}_\infty(f(x, t), f(x', t)) \leq \frac{\epsilon}{4}$. This means any point in B that can be reached by a trajectory of duration at most t_i is $\frac{\epsilon}{4}$ -close to a reachable point using an exact trajectory from $\llbracket R \rrbracket$.
- $\exists x : B \setminus \llbracket R \rrbracket \bullet \forall t : [0, t_i] \bullet f(x, t) \in \llbracket \mathbf{I} \rrbracket$. Use the current ϵ''_j and t_i in the formula we proved earlier and let ϵ''_{j+1} to be what returned by that formula. If $x \notin B_\infty^{\epsilon''_{j+1}}(\llbracket R \rrbracket)$ we don't need to care about it. Otherwise, there are two cases:
 - $\left(x \in B_\infty^{\epsilon''_{j+1}}(\llbracket R \rrbracket) \setminus \llbracket R \rrbracket \right)$. We know $\exists x' : \llbracket R \rrbracket \cap B_\infty^{\epsilon''_j}(x) \bullet \forall t' : [0, t_i] \bullet f(x', t') \in \llbracket \mathbf{I} \rrbracket$. x' is ϵ''_j -close to x . Therefore, the distance of x' and any point in B is bounded by $2\epsilon'$. This means any point that can be reached from B within t_i is $\frac{\epsilon}{2}$ -close to an

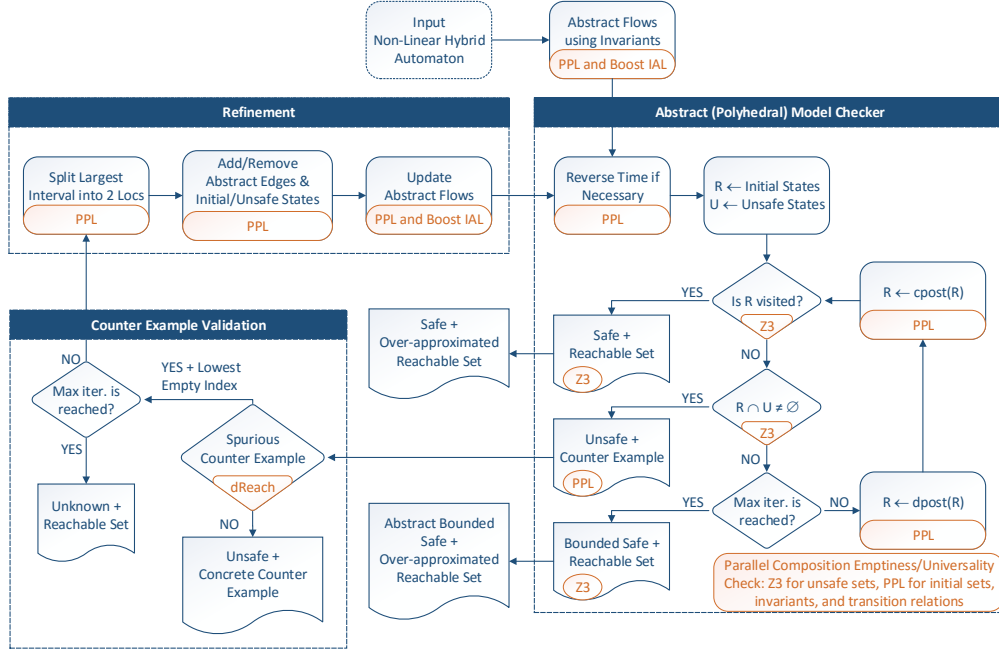


Figure 4.4: Flow Chart of HARE's CEGAR Loop

exact reachable point.

– $\left(x \notin B_{\infty}^{\epsilon''+1}(\llbracket R \rrbracket) \setminus \llbracket R \rrbracket \right)$. This means $x \in \llbracket R \rrbracket$ which is a contradiction.

It remains to prove that these three cases are in fact complementary. Formula $\forall \eta: \mathbb{R}_+ \bullet \exists x: B \bullet \forall t: [0, t_i] \bullet d_{\infty}(f(x, t), I) \leq \eta$ is the negation of the first condition. For any $n: \mathbb{N}_+$, let $\eta_n := \frac{1}{n}$. We have an infinite sequence of points x_n such that $\forall t: [0, t_i] \bullet d_{\infty}(f(x_n, t), I) \leq \frac{1}{n}$. Since B is bounded, it means there is a sequence $m_1 < m_2 < \dots$ such that x_{m_1}, x_{m_2}, \dots converges to a point x^* and for any $n: \mathbb{N}_+$ we have $\forall t: [0, t_i] \bullet d_{\infty}(f(x_{m_n}, t), I) \leq \frac{1}{m_n}$. Since B is closed, we know $x^* \in B$ and $\forall t: [0, t_i] \bullet d_{\infty}(f(x^*, t), I) = 0$. Since $\llbracket I \rrbracket$ is closed this is equivalent to $\forall t: [0, t_i] \bullet f(x^*, t) \in \llbracket I \rrbracket$, which means the disjunction of the last two cases is true.

□

4.3 Architecture of HARE

Figure 4.4 shows the flow and architecture of HARE. It also identifies 3rd party libraries/tools that are internally used by HARE at different steps. We use Z3 [70] to check if a fix-point is reached in the abstract system model-checking, and also to check whether an unsafe state is reached. We use Boost Interval Arithmetic Library (IAL) [71] to abstract non-linear dynamics. We use `dReach` to validate a counter-example (the validation a counter-example of length n involves at most n invocations of `dReach`). Note that `dReach` calls `dReal`, internally. Also, `dReach`/`dReal` are not available in the form of libraries. Therefore, HARE executes `dReach` as a separate process and communicates with it through files. Finally, we use Parma Polyhedra Library (PPL) [69] to manipulate symbolic abstract states. This includes, computing discrete/continuous abstract posts, constructing annotated counter examples, finding rectangular hull of a polytope, abstracting affine flows, and checking if a parallelly composed location/edge has non-empty invariant/transition relation. Compared to the old version of HARE in [65], we have replaced `SpaceEx` with `dReach`, since `SpaceEx` does not support non-linear dynamics. Also, we have implemented everything in C++ instead of Scala to improve performance.

The abstract model checker in HARE has a parameter `direction` with possible values `forward` and `backward`. It specifies whether the tool should perform forward or backward reachability. But PPL can only compute `cpost` and not `cpre`. This is the reason for the step “Reverse Time If Necessary”. There is an optional integer parameter `max-iter` for each of the abstract and concrete model checkers. If the maximum number of iterations is reached in the abstract model checker, it returns `bounded-safe` as an answer. If abstract model checker returns this answer to the concrete model checker, `abstract bounded safe` will be returned as a result. If the maximum number of iterations is reached in the concrete model checker, it returns `unknown` as the answer. In addition to `Safe` or `Unsafe`, the user can also ask HARE to produce a `counter-example`, an `annotated-counter-example`, or the `reachable-set`. Clearly, the first two will only be produced if the system is found to be unsafe and the last one will only output the *abstract* reachable states. Note that abstract model checker can be directly called by user.

The model to be checked along with all the options for the model checker

are specified in a single human readable text file according to `INFO Parser` from Boost Property Tree Library [74]. Every model, contains one or more hybrid automata and the safety problem is considered for their parallel composition which is constructed on the fly. Continuous variables can be read by all hybrid automata. If the file specifies polyhedral automata, each hybrid automaton can write to all variables through transition relations and flow. On the other hand, if the file specifies a non-linear polyhedral automaton, different hybrid automata can still write to a common variable through transition relations, but flow of a variable should be defined in exactly one hybrid automaton. Initial and unsafe states are specified after all hybrid automata using zero or more polyhedra for each composed location. Each edge has an optional label. If it is specified, it means that edge must be synced with an edge from other hybrid automata in the file. Otherwise, it will be interleaved. If a specified label does not end with ‘?’, ‘!’, or ‘!!’, synchronization will be among all hybrid automata in the file (*i.e.* each hybrid automaton must take an edge with the exact same label). Characters ‘?’, ‘!’, and ‘!!’ are used to specify input/output hybrid automata, where ‘?’ is for an input edge, ‘!’ is for an output edge, and ‘!!’ is for a broadcast edge. Character ‘*’ at the beginning of a location name means that location is transient and time cannot pass inside that location. Allowing transient locations in the model has three benefits 1. neither abstract nor concrete model checker will waste time by computing continuous post in transient locations, 2. the result automata will have one less variable, and 3. the model will be easier to understand. Finally, the current interface to the tool is only through the command line.

4.4 Experimental Results

The tool described in this chapter also appeared in [75], is a significant improvement over the version reported in [65]. First, the old version only verified affine hybrid automata. The new version also considers non-linear dynamics. Second, the old version used rectangular automata to abstract concrete models. The new version uses polyhedral hybrid automata. We have observed a marked improvement in running time due to the change in abstract models — there are fewer refinement iterations on many examples because of the use of polyhedral hybrid automata. Third, the tool has been

made robust. The implementation has migrated to C++ from Scala to improve its running time. We have changed some of the 3rd party tools that HARE uses internally. All these changes have enabled HARE to handle a larger class of examples (including more affine hybrid automata), with a faster running time. We also compare against the old version of HARE [65]. We show that the new tool successfully proves safety when the others fail.

The new version of HARE is available from <https://uofi.box.com/v/HARE>; the old version of the tool can be downloaded from <https://uofi.box.com/cegar-hare-tacas-2016>. Examples and scripts for running the examples can also be found on the links. Both these links contain a virtual machine to make repeatability straightforward.

We have run HARE with different set of examples with both affine and non-linear dynamics. Brief explanations of the affine benchmarks can be found in [65]. Table 4.1 contains the results for the affine examples. We compare the performance of HARE, its old version in [65], SpaceEx [42], PHAVer [43], and SpaceEx AGAR [44]⁶. The first two tools are affine hybrid automata model checkers that are not CEGAR based, while the last is a CEGAR based tool for concurrent hybrid automata⁷. In the past [65], we also reported the performance of HSolver [45] on affine examples. However, since it performed poorly on affine examples, we have not included it for comparison in Table 4.1.

Tank Benchmark [76]. Each problem in this benchmark consists of some $N \in \mathbb{N}$ tanks. Each tank $i \in \{1, \dots, N\}$ loses volume x_i at some constant flow rate v_i . Hence, dynamics of tank i is $\dot{x}_i = -v_i$ for a rational constant $v_i \geq 0$. Furthermore, one of the tanks is filled from an external inlet at some constant flow rate w which makes its dynamics $\dot{x}_i = w - v_i$, for a rational constant $w \geq 0$. The volume lost by each tank simply vanishes and does not move from one tank to another.

Satellite Benchmark [76]. These examples model two satellites orbiting the earth with nonlinear dynamics described by Kepler’s laws (see [77] for details). The nonlinear dynamics were hybridized in [76] to generate an affine hybrid automaton. The size of the problems varies from 36 to 1296 locations and so this benchmark can test the scalability of the tool. The

⁶By SpaceEx we mean SpaceEx with Supp as its scenario and by PHAVer we mean SpaceEx with PHAVer as its scenario.

⁷It is called “Assume Guarantee Abstraction Refinement” in [44].

Model	Dim.	Size	HARE						SpaceEx			PHaVer			SpaceEx AGAR			
			Time		Iters.		Safe		Time	FP.	Safe	Time	FP.	Safe	Merged Locs.	Time	FP.	Safe
			old	new	old	new	old	new										
Tank 16	3	3 / 6	< 1	< 1	1	1	✓	✓	3	✗	✗	1414	✗	✓	2	1133	✗	✓
Tank 17	3	3 / 6	< 1	< 1	1	1	✓	✓	5	✗	✓	1309	✗	✓	2	1041	✗	✓
Satellite 03	4	64 / 198	91	< 1	1	1	✗	✓	< 1	✗	✗	1804	✗	✗	28	> 600	---	---
Satellite 04	4	100 / 307	< 1	< 1	1	1	✓	✓	< 1	✗	✓	< 1	✓	✓	91	49	✓	✓
Satellite 11	4	576 / 1735	1	< 1	1	1	✓	✓	< 1	✗	✓	< 1	✓	✓	449	> 600	---	---
Satellite 15	4	1296 / 3895	2	< 1	1	1	✓	✓	< 1	✗	✓	< 1	✓	✓	264	> 600	---	---
Heater 03	3	4 / 6	> 600	54	---	1	---	✓	84	✗	✗	< 1	✓	✗	---	---	---	---
Heater 05	3	4 / 6	< 1	58	1	38	✗	✓	61	✗	✓	< 1	✓	✗	---	---	---	---
Heater 09	3	4 / 6	< 1	80	1	15	✗	✓	42	✗	✗	< 1	✓	✗	---	---	---	---
Nav 01	4	25 / 80	9	18	11	11	✓	✓	< 1	✓	✓	< 1	✓	✓	21	5	✓	✓
Nav 08	4	16 / 48	7	< 1	13	1	✓	✓	685	✗	✓	< 1	✓	✓	10	< 1	✓	✓
Nav 09	4	16 / 48	7	< 1	10	1	✓	✓	< 1	✗	✗	< 1	✓	✗	4	< 1	✓	✗
Nav 13	4	9 / 18	8	< 1	15	1	✓	✓	< 1	✗	✓	< 1	✓	✓	4	< 1	✓	✓
Nav 19	4	33 / 97	29	< 1	17	1	✓	✓	2	✗	✓	< 1	✓	✓	11	< 1	✓	✓

Table 4.1: Comparing HARE with its old version in [65] and other tools for affine dynamics. Dim. is the number of continuous variables. Size is the number of locations/edges in the input (concrete) model. Iters. is the number of iterations in our CEGAR loop before proving safety. FP. tells whether or not a tool reached a fixed-point. If a tool does not reach a fixed-point then even if it says the system is safe, the answer may not be true. As explained in [65], sometimes SpaceEx tells it reached a fixed-point, but before that it generates a warning that its result may not be complete. We continue to consider those cases as SpaceEx has not reached a fixed-point. Merged Locs. is the number of locations we initially merged for SpaceEx AGAR. Columns old and new for HARE contain results from the previous and current version of this tool. All times are in seconds and all examples were run on a laptop with Intel i5 2.50GHz CPU and 6GB of RAM.

safety property being checked is collision avoidance, *i.e.*, whether there is a trajectory in which satellites come too much close to each other.

Heater Benchmark [76]. There are three rooms with three heaters. For each room, we have one automaton with two states modeling heater being on and off in that room. Composition of these three room automata gives us a heater system.

Navigation Benchmark [22, 78]. This benchmark considers a robot moving in the \mathbb{R}^2 plane. There is a desired velocity v_d that is determined by the current location of the object in an $n \times m$ grid. Each grid has one of the 8 possible desired velocities pointing to the usual 8 possible directions in the plane. Dynamics of object’s velocity is determined by $\dot{v} = \mathcal{A}(v - v_d)$ where $\mathcal{A} \in \mathbb{Q}^2$. There are two special type of cells. Those that are unsafe and those that are blocked. Some of the problems in this class use the following variation: For an small value $\epsilon \in \mathbb{Q}_+$, neighbor cells overlap with each other. This introduces non-nondeterminism into the model.

The new version of HARE proved all examples are safe, while the old version

could not do this for four examples. Also the new version is faster on all examples, except one. **SpaceEx** almost never reached a fixed-point. **PHAVer** could prove safety for only half of the examples, and it did it faster than new version of **HARE** in only one case. Abstraction in **SpaceEx AGAR** appears to be a very expensive operation — in four examples, the initial abstraction was not constructed even after 600 seconds (10 minutes) and we terminated the execution. Also, in three examples we could not find any set of locations that does not cause the tool to crash right at the beginning. Among 8 examples that worked for **SpaceEx AGAR**, it could prove safety for 5 of them and it was always slower than new version of **HARE**.

Table 4.2 contains results of comparing **HARE** with **C2E2** [46], **HSolver** [45], and **FLOW*** [47] on nonlinear examples. Note that **HARE** and **HSolver** support proving safety for unbounded time and unbounded number of discrete transitions. But both **C2E2** and **FLOW*** require bounded time and bounded number of discrete transitions. Also none of these two tools check whether the computed (unbounded) reachable set so far is a fixed-point. Therefore, no matter how big the time-bound is set, proving safety for this time bound in these tools does not guarantee unbounded time safety. In our experience, we set the bound for discrete number of transitions large enough so none of the tools terminated because of this bound being reached. For the first 5 examples, we set the time bound equal to 1000 in **C2E2** and **HSolver**. For the last example, the time bound is 10 in all tools. **HARE** always finished faster than **C2E2**. On three examples **HARE** is faster than **FLOW*** and only in one example is slower. On 3 examples **HARE** proved safety faster than **HSolver**, and in 2 examples **HSolver** was faster. **HSolver** comes with an example named **circuit** (not reported in Table 4.2). The size of hybrid automaton in this example is small, but it has constants of the order 10^{12} , which turns out to be too big for **C2E2** and **dReach** and trigger a bug in these two tools (and hence **HARE**). Only **HSolver** proves safety of this example. Finally, in our experiments, **dReach** performs much faster for the affine dynamics. Non-linear examples are also available at link for the new version of **HARE** we mentioned earlier.

Model	Dim.	Size	HARE		Time Bound	C2E2	HSolver	FLOW*
			Reached Abst. Size	Time		Time	Time	Time
Van der Pol	2	1 / 0	26 / 194	∞	< 1	56	3*	> 600
Jet Engine	2	1 / 0	189 / 1330	∞	55	56	2*	> 600
Cardiac Cell	2	2 / 2	249 / 1783	∞	16	50	< 1*	25
Cardiac Control	3	2 / 2	270 / 3974	∞	153	> 600	> 600*	41
Clock	3	1 / 0	9 / 56	∞	< 1	---	< 1	< 1
Sinusoid	2	1 / 0	32 / 62	10	< 1	1	7	---

Table 4.2: Comparing running time of HARE with other tools for non-linear dynamics. Dim. is the number of continuous variables. Size is the number of locations/edges in the input (concrete) model. Reached Abst. Size is the number of locations/edges in the final abstract model that are reached in HARE right before safety is proved. Time Bound is 10 for the “Sinusoid” model in all four tools. For all the other examples, there is no time bound in both HARE and HSolver. In other word, HARE and HSolver prove unbounded time safety for all but the last example. C2E2 and FLOW* on the other hand, require finite time bound, and we set it to be 1000 (except for the “Sinusoid” model which is 10). We have terminated all the runs that took more than 600 seconds (10 minutes). HSolver requires bounded invariants. So in the first four examples, we put 100 as an upper bound and -100 for as a lower bound of unbounded variables. FLOW* does not support trigonometric functions and C2E2 encounters an internal error on one of the examples. All times are in seconds and all examples were run on a laptop with Intel i5 2.50GHz CPU and 6GB of RAM.

4.4.1 Unbounded Invariants

The first 4 examples in Table 4.2 are taken from C2E2. Tools like C2E2 and FLOW* that try to compute the reachable set as precisely as possible, tend not to specify invariants. On the other hand, tools like HARE and HSolver that perform refinement by partitioning the state space tend to require bounded invariants. Another reason for HARE to prefer bounded invariants is that dReach, which HARE uses internally, only works for bounded variables. We had a few options to bound the invariants in those examples. The first option is to bound the invariants using large enough numbers (just like what we did for HSolver). This means we are guessing the invariant. If the guessed invariants are all closed sets, one can verify the guess by setting closure of complement of it as the unsafe states. If the unsafe states are not reachable then the guess is valid. Note that since HARE computes an over-approximation of unsafe states, it is possible that HARE incorrectly says a guessed invariant is invalid. The second option is to first use tools like C2E2 or FLOW* and find a coarse invariant for all locations. Note that since these tools have bounded number of discrete transitions and they do not check for fixed-point, one

might still need to verify that invariants obtained using **C2E2** or **FLOW*** are valid. The third option, which we have used for the current implementation, is noticing that the only part of the implementation that requires invariant to be bounded is where **dReach** is called. If this tool is called with an unbounded variable, then it will quickly raise an exception and terminate. In other words, it will terminate without saying that the counter-example is valid. We take *not saying valid* as saying *invalid*. This approach makes it possible to use **dReach** even when invariants are not bounded. Note that during validation of a counter-example of length larger than one, it is possible that only invariants after some step k are unbounded. Our current approach guarantees all variables are bounded when **dReach** is called for indices k or smaller. An example of such a system, *Automatic lane change system (driver assist)* that comes with **C2E2**. It is a system with affine dynamics and 10 unsafe sets. **HARE** proved unbounded safety for all these sets in about 190 seconds. During this time, **dReach** encountered exception in almost every iteration. But eventually, the abstract model checker reached a fixed-point and found the system to be safe, so **dReach** was not called again. **C2E2** needs to prove safety for each of these sets separately and it took this tool about 1163 seconds to prove them all when the time bound is set to 1000. **HSolver** and **FLOW*** could not prove safety for any of these sets within 600 seconds (10 minutes)⁸. A fourth option is one where we initially partition the state space blindly for a small number of times first, and then start the actual CEGAR loop. We used this option in all four examples in Table 4.2 from **C2E2**.

4.5 Related Work

Doyen *et al.* considered rectangular abstractions for safety verification of affine hybrid systems in [79]. However, their refinement is not guided by counter-example analysis. Instead, a reachable unsafe location in the abstract system is determined, and the invariant of the corresponding concrete location is split to ensure certain optimality criteria on the resulting rectangular dynamics. This, in general, may not lead to abstract counter-example elimination, as in our CEGAR algorithm. We believe that the refinement

⁸Time bound for **HSolver** and **HARE** are set to be the same. Similarly, time bound for **FLOW*** and **C2E2** are set to be the same.

algorithms [79] and the one in this chapter are incomparable — one may perform better than the other on certain examples. Empirical evaluations could provide some insights into the merits of the approaches, however, the implementation of the algorithm in [79] was not available for comparison at the time of writing this thesis.

Bogomolov *et al.* consider polyhedral inclusion dynamics as abstract models of affine hybrid systems for CEGAR in [44]. Their abstraction merges the locations, and refinement corresponds to splitting the locations. Hence, the CEGAR loop ends with the original automaton in a finite number of steps, if safety is not proved by then. Our algorithm in this chapter, splits the invariants of the locations, and hence, explores finer abstractions. Our method is orthogonal to that of [44], and can be used in conjunction with [44] to further refine the abstractions.

Nellen *et al.* use CEGAR in [80] to model check chemical plants controlled by programmable logic controllers. They assume that the dynamics of the system in each location is given by *conditional* ODEs, and their abstraction consists of choosing a subset of these conditional ODEs. The refinement consists of adding some of these conditional ODEs based on an unsafe location in a counter-example. The method does not ensure counter-example elimination in successive iterations. Their prototype tool does not automate the refinement step, in that the inputs to the refinements need to be given manually. Hence, we did not experimentally compare with this tool.

Zutshi *et al.* propose a CEGAR-based search in [81] to find violations of safety properties. Here they consider the problem of finding a concrete counter-example and use CEGAR to guide the search of the same. We instead use CEGAR to prove safety — the absence of such concrete counter-examples.

4.6 Conclusions

We presented a new algorithm for model checking safety problems of affine hybrid automata in a counter-example guided abstraction refinement framework. We show that our algorithm is sound and have implemented it in a tool named **HARE**. We also compared the performance of our tool with a few state-of-the-art tools. Results show that performance of our tool is promising

compared to the other tools (**SpaceEx**, **PHAVer**, and **HSolver**).

In the future, we intend to incorporate certain improvements to our implementation. In particular, we would like to integrate an algorithm for computing an under-approximation of the continuous post. This will allow us to definitively validate abstract counter-examples. Theoretically, we would like to explore the completeness of our algorithm, in terms of finding a concrete counter-example when the concrete system is unsafe. This may require a novel notion of counter-example in the abstract system, which is shortest in terms of the number of edges in the concrete system which do not correspond to self-loops.

Chapter 5

Robust Model Checking

A closer look at our undecidability result in Chapter 3, reveals that it is relying on possibility of storing unbounded amount of information in real values between any two consecutive natural numbers. In other words, it uses an encoding/decoding of all natural numbers into a non-empty interval (a, b) . As a result, if we perturb the dynamics of the automaton we constructed in Section 3.1, the result automaton might not be able to simulate 2-counter machine anymore. Our results in Chapter 4 gives us another motivation for robust verification. We proved there that if unsafe and reachable sets of states are separated, our algorithm can always prove the safety of the automaton. In other words, if the automaton is safe, but the unsafe and reachable sets are too close, our algorithm might not terminate or it might even say the system is unsafe.

Looking at the process of model checking cyber-physical systems “in general” reveals the same problem. A lot of undecidability proofs in the literature rely on the non-existence of noise, meaning that they assume that the sensors are perfect and there is no noise in the physical environment. Furthermore, a lot of designs satisfy the required properties by assuming the absence of noise in the environment. Existence of this unrealizable assumption in design and verification, leads one to question the modeling language used to specify different classes of hybrid automata and their problems. *Robust* model checking aims to address these problems that is suggested by researchers in [30–41].

In this chapter, we consider robust model checking of timed automata. Timed automata [11] are the standard formal model for real-time systems because they are an elegant and expressive formalism, and, yet are amenable to algorithmic analysis [82, 83]. However, timed automata, as many other classes of hybrid automata, have an idealistic semantics that makes assumptions that are physically unrealizable. To be more concrete, in timed automata, time is measured by perfectly continuous and synchronous clocks

that have infinite precision as opposed to finite precision, almost synchronized, digital clocks that are accessed by implementations. Timed automata can also respond instantaneously to events while physical realizations of the real-time systems react with some non-zero delay. Finally, timed automata allow modelling control algorithms that exhibit Zeno behaviors, have unrealistic convergence properties [31], or isolated behaviors [30]. These deficiencies have been observed by a number of researchers [30–35]. To ensure that correct timed automata yield correct, implementable designs, the remedy suggested by these papers is to consider a *robust* semantics for timed automata.

Starting from the seminal work of [30] where a topological notion of robustness was proposed, different notions of robustness have been considered [33, 35, 84]. One notion of robustness that has been studied extensively, is the one that is formally defined in Section 2.2.3. This notion of robustness has been shown to imply implementability in real-time platforms [33, 85].

The algorithmic complexity of checking the robustness of timed automata designs has received much attention. For a timed automaton \mathcal{H} , remember that by \mathcal{H}^ϵ , we denote the semantics where clocks can drift by ϵ (but guards remain unperturbed), and by \mathcal{H}_δ , we denote the one where guards are enlarged by δ (but clocks do not drift). Robust safety/reachability problem is defined in Problem 6 and robust ω -regular model checking is defined in Problem 7. The robust safety problem was first considered in [84]. It is shown that robustness with only clocks drifts with respect to safety properties can be decided in PSPACE. These results were generalized in [85] where safety verification under both clock drifts and enlarged guards is solved in PSPACE. The approach in both [84] and [85] is based on computing the reachable states under infinitesimal guard and clock perturbations (or only guard or only clock perturbations), referred to as $\text{limreach}_\delta^\epsilon(\mathcal{H})$ (or $\text{limreach}_\delta(\mathcal{H})$ or $\text{limreach}^\epsilon(\mathcal{H})$). It is shown that \mathcal{H} is robustly safe (under just clock drifts, or guard perturbations, or both) iff the corresponding limreach set is disjoint from the unsafe states. Moreover, it is observed in [85] that the three limreach sets coincide, that is,

$$\text{limreach}_\delta^\epsilon(\mathcal{H}) = \text{limreach}^\epsilon(\mathcal{H}) = \text{limreach}_\delta(\mathcal{H}) \quad (5.1)$$

Except the results about disjointness of unsafe and limreach sets, all the above results on safety verification (including the fact that limreach sets coincide)

in [84, 85] are established for timed automata that satisfy the *progress cycle assumption (PCA)* which requires that every cycle in the region graph of \mathcal{H} reset every clock of timed automaton at least once.

The progress cycle assumption can be restrictive when modeling real-time systems because it does not allow the design to measure time spent in cycles. Therefore, the decidability of robust verification without the progress cycle assumption was investigated in [86]. The results from [84, 85] were generalized in a couple of directions. First the progress cycle assumption was removed. Second, general ω -regular properties were considered as opposed to just safety. In addition, the restriction to bounded timed automata was also removed; this restriction to bounded automata is, however, not limiting because every timed automaton is weakly bisimilar to a bounded timed automaton [87, 88]. However, in some respects, the results in [86] are also less general than those in [85]; the paper [86] only considers robustness with respect to perturbation of guards alone.

In this chapter, we continue this line of investigation. We first consider the problem of robust safety model checking of monotonic rectangular automata when only clocks are drifted and show that it is NEXPTIME-complete. This will be used in many of our results on timed automata. In the absence of the progress cycle assumption, we then ask what is the complexity of robustly verifying a property when guards are enlarged, as well. Our first observation is that an automaton \mathcal{H} robustly satisfies a property B when both clocks and guards are perturbed iff \mathcal{H} robustly satisfies the same property when only guards are enlarged. In addition, $\text{limreach}_\delta^\epsilon(\mathcal{H}) = \text{limreach}_\delta(\mathcal{H})$. Thus, using the algorithm in [86], one can verify designs under both clock drifts and guard perturbation. On the other hand, we show that robustness when only clocks are drifted is not equivalent with the *stronger* notion of robustness when both guards and clocks are perturbed. More precisely, we show that there are timed automata that are robust when only clocks are drifted, but not when both guards and clocks are perturbed (see Example 47). This contrasts with Equation (5.1) that holds under the progress cycle assumption. We then present an algorithm to check ω -regular properties when only clocks are drifted. We show that a timed automaton \mathcal{H} robustly satisfies a property B when only clocks are drifted iff there is a constant δ_1 (depending only on the size of \mathcal{H}) such that the automaton in which only the guard constraints involving *positive* constants are perturbed by δ_1 , satisfies B . This observation

Problem / Assumption	With PCA	Without PCA
$\text{limreach}_\delta^\epsilon$ computation	✓ [85]	
limreach_δ computation	✓ [85]	
limreach^ϵ computation	✓ [84, 85]	(✓)
Robust ω -regular	(✓)	(✓)
Robust ω -regular with only Enlarged Guards	✓ [89]	✓ [86]
Robust ω -regular with only Drifted Clocks	(✓)	(✓)

Table 5.1: Summary of Robust Model Checking Problem Results: Check marks indicate that the corresponding problem is solvable in polynomial space; check marks within parenthesis are established in this chapter and those without the parenthesis were established in the reference cited alongside. Recall that PCA refers to the Progress Cycle Assumption.

can be exploited to give a PSPACE algorithm. We also show that this problem is PSPACE-hard, thus establishing the optimality of our algorithm.

Next, we consider the problem of computing limreach^ϵ for bounded timed automata in the absence of the progress cycle assumption. While the algorithm to verify ω -regular properties discussed in the previous paragraph also applies to robust safety verification, computing limreach^ϵ is of independent interest just like computing reachable sets is an important task independent of safety verification. Puri’s algorithm [84] (generalized in [85]) works by iteratively adding the (topological) closure of regions on progress cycles that have a non-empty intersection with the current limreach set. We show that the (almost) same algorithm correctly computes limreach^ϵ even when the progress cycle assumption does not hold. This algorithm can be shown to use polynomial space. Our contributions, in the context of earlier results, is summarized in Table 5.1. As can be seen from the table, the results in this chapter cannot be used to compute $\text{limreach}_\delta^\epsilon$ nor limreach_δ , in the absence of the progress cycle assumption.

5.1 Bounded Time Robust Reachability

In this section, we prove that bounded time robust reachability of monotonic rectangular automata is NEXPTIME-complete. Problem 34 formalizes what we are considering in this section. In Section 5.1.1 we briefly review the proof for

the non-robust version of the problem (*i.e.* $\epsilon = 0$ in Problem 34) and formally extend it to the robust version. In Section 5.1.2 we present our algorithm for solving this problem in NEXPTIME. Finally, in Section 5.1.3 we prove the problem is NEXPTIME-hard, which implies that our algorithm is optimal.

In this section, for a timed automaton \mathcal{H} , a *general zone* is a finite disjunction of finite conjunction of constraints of the form $x - y \bowtie c$ or $x \bowtie c$, where x, y are variables of \mathcal{H} , c is an integer, and \bowtie is among $\{<, \leq, =, \geq, >\}$.

Problem 34. Given a monotonic rectangular automaton \mathcal{H} , a time bound $T : \mathbb{N}$, and an unsafe general zone Z in \mathcal{H} , is it true that for all $\epsilon : \mathbb{R}_+$, Z can be reached in \mathcal{H}^ϵ within T units of times? That is whether or not the following property holds?

$$\forall \epsilon : \mathbb{R}_+ \bullet \exists \tau : \llbracket \mathcal{H}^\epsilon \rrbracket_* \bullet \text{first}(\tau) \in \mathbf{S}^{\text{init}}_{\llbracket \mathcal{H} \rrbracket} \wedge \text{last}(\tau) \in Z \wedge \text{duration}(\tau) \leq T$$

5.1.1 From Bounded Time to Bounded Length

Brihaye *et al.* solved the non-robust version of Problem 34 in [16] (*i.e.* $\epsilon = 0$). The main observation in [16] is that if there is a trajectory $\tau : \llbracket \mathcal{H} \rrbracket_*$ from state (q_1, ν_1) to (q_2, ν_2) such that $\text{duration}(\tau) \leq T$ then there is a trajectory τ' from (q_1, ν_1) to (q_2, ν_2) whose length is at most exponential in the size of \mathcal{H} and linear in T . Furthermore, τ' has the same duration and visits the exact same set of locations as τ . Theorem 35 formalizes their result. Note that properties 3 and 4 in this theorem are not explicitly mentioned in the body of Theorem 2 in [16], but they are obvious from the proof.

Theorem 35 (Theorem 2 in [16]). Let \mathcal{H} be a monotonic rectangular automaton in which all constants are integers and $T : \mathbb{R}_{\geq 0}$ be a time bound. For any $\tau : \llbracket \mathcal{H}^\epsilon \rrbracket_*$ there is $\tau' : \llbracket \mathcal{H}^\epsilon \rrbracket_*$ with the following properties:

1. $\text{first}(\tau') = \text{first}(\tau)$,
2. $\text{last}(\tau') = \text{last}(\tau)$,
3. $\text{edges}(\tau') = \text{edges}(\tau)$,
4. $\text{duration}(\tau') = \text{duration}(\tau)$, and
5. $|\tau'| \leq F(\mathcal{H}, T) = 24(T \times \text{rmax} + 1) \times |\mathbf{X}|^2 \times |\mathbf{Q}|^2 \times (2\text{cmax} + 3)^{2|\mathbf{X}|}$.

Where rmax is the maximum possible rate (in absolute value) of a variable in \mathcal{H} , and cmax is the maximum constant (also in absolute value) in specification of \mathcal{H} .

Function $F(\mathcal{H}, T)$ in Theorem 35 assumes constants in guards are integers. So the minimum distance between distinct constants is at least 1. But if we allow rational constants in guards, the minimum distant is at least $\frac{1}{2^n}$ where n is the size of \mathcal{H} , assuming constants are given in binary format. This increases value $F(\mathcal{H}, T)$ but it will remain exponential in n and linear in T . Finally, as far as Theorem 35 and its proof are concerned, flows of variables could be rational as well. All we need from the flows is that they should be monotone and bounded by rmax , two properties that are true by definition.

Theorem 36. Let \mathcal{H} be a monotonic rectangular automaton, $\epsilon : [0, \text{rmax}] \cap \mathbb{Q}$ be a perturbation, and $T : \mathbb{R}_{\geq 0}$ be a time bound. Then for any $\tau : \llbracket \mathcal{H}^\epsilon \rrbracket_*$ there is $\tau' : \llbracket \mathcal{H} \rrbracket_*$ with the following properties:

1. $\text{first}(\tau') = \text{first}(\tau)$,
2. $\text{last}(\tau') = \text{last}(\tau)$,
3. $\text{edges}(\tau') = \text{edges}(\tau)$,
4. $\text{duration}(\tau') = \text{duration}(\tau)$, and
5. $|\tau'| \leq F(\mathcal{H}, 2T)$ (F is defined in Theorem 35).

Proof. For any given $\epsilon : [0, \text{rmax}] \cap \mathbb{Q}$, \mathcal{H}^ϵ is also a monotonic rectangular automaton. Furthermore, $\text{rmax}_{\mathcal{H}^\epsilon} \leq 2\text{rmax}_{\mathcal{H}}$. Also, \mathcal{H} and \mathcal{H}^ϵ have the exact same components except their flows. Using Theorem 35, the proof is complete once we notice $F(\mathcal{H}^\epsilon, T) \leq F(\mathcal{H}, 2T)$. \square

5.1.2 NEXPTIME Algorithm

Knowing that $F(\mathcal{H}, T)$ is a bound on the maximum length of a set of trajectories that witness the set of states reachable within T units of time, the algorithm in [16] and its correctness are quite straightforward. It first guesses a path π of length at most $F(\mathcal{H}, T)$ and then constructs conjunction of the

set of linear formulas φ that are satisfiable iff π is an induced path of a witness trajectory. The algorithm then uses linear programming to check, in exponential time, whether or not φ is satisfiable.

For us the length of path π is bounded by $n := F(\mathcal{H}, 2T)$. The generated formula φ is of the form $\forall \epsilon : \mathbb{R}_+ \bullet \exists t_0, \dots, t_n : \mathbb{R}_+ \bullet \exists \nu_1, \dots, \nu_n : \mathbf{V} \bullet \exists \nu'_0, \dots, \nu'_n : \mathbf{V} \bullet \psi$ in which $n' \leq n$ is the number of non-zero continuous transitions, t_i 's are non-zero durations of continuous transitions, ν_i 's are states reached after continuous transitions, and ν'_0 is an initial state, and ν'_i 's (for $i > 0$) are states reached after discrete transitions ¹ (see Section 2.2.2 for the formal definition of trajectories and executions). In general, one can consider one variable t_i for the i^{th} continuous transition, but then we should say these variables are non-negative numbers as opposed to positive numbers. The assumption that time variables are all positive, will be used in our proof. Note that the form we chose, requires the algorithm to non-deterministically select non-zero variables.

Using Definition 3, ψ is a conjunction of a set of formulas of the following forms ($u : \mathbf{X} \rightarrow \mathbb{Q}$ is a vector, $u \cdot \nu$ is the dot/inner product of u and ν , $c : \mathbb{Q}$ is an arbitrary rational number, $\triangleleft : \{<, \leq\}$ is the comparison type, and absolute value of all constants is bounded by \mathbf{cmax} or \mathbf{rmax}):

1. $0 < t_i$,
2. $t_0 + t_1 + \dots + t_n \triangleleft T$,
3. $u \cdot \nu_i \triangleleft c$ or $u \cdot \nu'_i \triangleleft c$,
4. $\nu_{i+1}^x - \nu_i^{x'} \triangleleft ct_i + \epsilon t_i$ or $\nu_i^{x'} - \nu_{i+1}^x \triangleleft ct_i + \epsilon t_i$

Formulas of type 1 enforce that values of selected non-zero durations are indeed positive. Formula of type 2 enforces the time bound. Formulas of type 3 enforce invariants and transition relations. Formulas of type 4 enforce relation between old and new values of variables after continuous transitions.

We cannot take the same approach as in [16] for three reasons: 1. Although the quantifier free first order theory of rationals with addition (which is used in [16]), is known to be decidable in polynomial time [90], the full theory is known to be between single and double exponential in time [91]. If we want

¹Each ν_i 's and ν'_i are actually $|\mathbf{X}|$ independent variables of type \mathbb{R} . Therefore, φ is a quantified first order formula.

Algorithm 2 Robust Reachability Analysis of Monotonic Rectangular Automata

Input: Monotonic rectangular automaton \mathcal{H}

Input: General unsafe zone Z

Input: Time bound T

Output: Answer to Problem 34

1. Guess a path π of length at most $F(\mathcal{H}, 2T)$.
 2. Guess variables t_i that are not 0
 3. $\psi \leftarrow$ the set of constraints representing reachable set after π
 4. $\psi_1 \leftarrow$ the set of type 4 formulas in ψ with non-zero t_i
 5. $\psi_2 \leftarrow \psi \setminus \psi_1$
 6. $\bar{\psi}_1 \leftarrow$ closure of ψ_1
 7. **return** $\bar{\psi}_1 \cap \psi_2 \neq \emptyset$
-

to use the full theory (since we have both quantifiers), the result would be an algorithm with a complexity between double and triple exponential time (ψ is exponentially big). 2. Formulas of type 4 are not even linear. Therefore, we need to use the full first order theory of rationals with addition and multiplication which is known to be undecidable [92]. 3. Of course we can use the decidable full theory of reals with addition and multiplication. But since this class of formula has EXPSPACE complexity [93] and our formula is of exponential size, this approach results in a double-EXPSPACE algorithm.

Algorithm 2 shows our approach to decide Problem 34 in NEXPTIME. At line 1 we guess a path π of the bounded length exactly like [16]. At line 2 we guess the set of variables t_i whose values must be positive. At line 3 we construct the quantifier free formula ψ similar to [16]. The difference is that ψ contains only those time variables that are guessed to be positive. Note that ψ is created using \mathcal{H} and not \mathcal{H}^ϵ . So it has no ϵ in it. At line 4 we let ψ_1 to be the set of constraints of type 4 with non-zero time variables. At line 5 we let ψ_2 to be the set of constraints that are in ψ but not ψ_1 . Line 6 defines closure of ψ_1 ². Finally, at line 7 we return **yes** iff the given condition is true. Note that, ψ_2 , and $\bar{\psi}_1$ each define a convex polyhedron. Therefore they are closed under intersection and we can check, in exponential time (because they are defined by exponentially many constraints), whether or not their intersection is empty.

²Closure is constructed by replacing all strict inequalities with non-strict inequalities.

Theorem 37. Algorithm 2 always returns the right answer.

Proof. Let k be the number of variables in ψ and $\epsilon: \mathbb{R}_+$ be an arbitrary number. Also, let ψ_1^ϵ be same as ψ_1 except it has ϵt_i factors back in it (recall that ψ_1 is the set of formulas of type 4). There are two cases:

- $\bar{\psi}_1 \cap \psi_2 \neq \emptyset$: Pick any point p in this set and let $t := \min\{p(t_0), \dots, p(t_{n'})\}$ ³, $m := k \max\{\text{rmax}, \text{cmax}\}$, and $\alpha := \frac{\epsilon t}{2m}$. Note that t is positive. Value $m\alpha$ is an upper bound on the change in a linear term $u \cdot \nu$, when value of every variable in ν is perturbed by at most α . $B_\infty(p, \alpha) \subseteq \psi_1^\epsilon$, and since $d_\infty(p, \psi_2) = 0$ we know $B_\infty(p, \alpha) \cap \psi_2 \neq \emptyset$. Therefore, $\psi_1^\epsilon \cap \psi_2 \neq \emptyset$ which implies **yes** is the correct answer to the input problem.
- $\bar{\psi}_1 \cap \psi_2 = \emptyset$: For any $p: \mathbb{R}^k$ there are two sub-cases:
 - $p \notin \psi_2$: Since ψ_2 represents the set of constraints in ψ that perturbation won't affect them, using the guessed path and positive times, p cannot be reached in \mathcal{H}^ϵ within T units of time (for any $\epsilon: \mathbb{R}_{\geq 0}$).
 - $p \in \psi_2$ and $p \notin \bar{\psi}_1$: Since $\bar{\psi}_1$ is a closed set, $d_\infty(p, \bar{\psi}_1) > 0$. Therefore, we know for some $\delta: \mathbb{R}_+$ and a type 4 formula $x - y \triangleleft ct + \epsilon t$ we have $p(x) - p(y) > cp(t) + \delta$ (otherwise $p \in \bar{\psi}_1$). Let ϵ be any value in $[0, \frac{\delta}{2 \max_i p(t_i)}]$. Clearly, p does not satisfy $x - y \leq ct + \epsilon t$. Thus, $p \notin \psi_1^\epsilon$. Therefore, the correct answer to the problem is **no**.

□

Corollary 38. Time bounded robust reachability problem for monotonic rectangular automata with only perturbed flows is in NEXPTIME.

5.1.3 NEXPTIME-hardness

In this section, we follow the footsteps of Brihaye *et al.*'s paper and reduce the membership problem of non-deterministic exponential time Turing machines to the *robust* time bounded reachability for monotonic rectangular

³For any variable x and point p , $p(x)$ is the value of x at p

automata. Brihaye *et al.* proved that the non-robust version of the Problem 34 is NEXPTIME-hard (*i.e.* when ϵ is set to 0). But since their gadgets for simulation are not robust to any perturbation, they cannot be directly applied here. For example, in their construction, in order to check a bit is zero $y \leq \frac{1}{2} \wedge x < \frac{1}{2}$ is used (for some well-defined variables x and y), and in order to check a bit is one $y \leq \frac{1}{2} \wedge x \geq \frac{1}{2}$ is used. This means, if $x = \frac{1}{2}$ then we conclude the bit is 1, but any smaller value gives us different conclusion.

Each state of the input Turing machine \mathcal{A} is encoded as (q, w_0, w_1) where q is a location of \mathcal{A} and w_i , for $i \in \{0, 1\}$, are two stacks of 0's and 1's, representing the only tape of \mathcal{A} . The following operations are sufficient to simulate \mathcal{A} :

- emptiness check of a stack,
- read top of a non-empty stack,
- push 0 or 1 into a stack, and
- pop 0 or 1 from a non-empty stack.

Note that each step of \mathcal{A} can be modeled using some tests plus at most one push and one pop. A stack $w = w_0 w_1 \dots w_{|w|}$, with w_0 being the top, is represented by a pair of numbers l and c . Value of $l := \frac{1}{4^{|w|}}$ represents size of w , and value of $c := \sum_{0 \leq n < |w|} \frac{w_n}{4^{n+1}}$ represents elements of w . Note that since the hybrid automaton will be perturbed, we cannot expect l and c to have their exact values. We will prove that if the perturbation is small enough, using our operations, one can still use l and c to precisely construct w .

Emptiness and top of a stack are checked using the following guards on discrete edges:

- $|w| = 0$ iff $l \geq \frac{5}{8}$ (note that $\frac{5}{8}$ is strictly larger than $\frac{1}{4}$),
- $|w| > 0$ iff $l \leq \frac{5}{8}$ (note that $\frac{5}{8}$ is strictly smaller than 1),
- $w_0 = 0$ iff $l \leq \frac{5}{8} \wedge c \leq \frac{1}{6}$ (note that $\frac{5}{8}$ is strictly smaller than 1 and $\frac{1}{6}$ is strictly smaller than $\frac{1}{4}$),
- $w_0 = 1$ iff $l \leq \frac{5}{8} \wedge c \geq \frac{1}{6}$ (note that $\frac{5}{8}$ is strictly smaller than 1 and $\frac{1}{6}$ is strictly larger than $\frac{1}{12} = \sum_{n \in \mathbb{N}} \frac{1}{4^{n+2}}$).

push and pop operations are defined using the following expressions on l and c :

- $\text{push}(w, 0) := l \leftarrow \frac{l}{4}$ and $c \leftarrow \frac{c}{4}$,
- $\text{push}(w, 1) := l \leftarrow \frac{l}{4}$ and $c \leftarrow \frac{c}{4} + \frac{1}{4}$,
- $\text{pop}(w, 0) := l \leftarrow 4l$ and $c \leftarrow 4c$,
- $\text{pop}(w, 1) := l \leftarrow 4l$ and $c \leftarrow 4(c - \frac{1}{4})$.

Figure 5.1 show different gadgets for multiplication and division by 4, and addition and subtraction by $\frac{1}{4}$. z is a helper variable and x is a member of $\{l_1, l_2, c_1, c_2\}$. Variables $\{l_i, c_i\}$, for $i: \{1, 2\}$, are used to simulate the stack w_i . When any of the variables in $\{l_1, l_2, c_1, c_2\}$ is updated, all other variables are constant (after perturbation their dynamics will be $[0, \epsilon]$). In case of no perturbation, x is at most 1. Assume that for a small enough ϵ , absolute value of x is bounded by 2. Figure 5.1 shows that after execution of each gadget the absolute error will be increased by at most 20ϵ ⁴. Since each instruction of \mathcal{A} requires at most 3 different gadgets, after each instruction the absolute error is increased by at most $4(4(4x + 20\epsilon) + 20\epsilon) + 20\epsilon - 64x = 420\epsilon$, which can only happen if we have 3 multiplications. Assume \mathcal{A} stops within N number of steps (we know N is exponential in size of \mathcal{A} , but our robustness argument does not depend on that). We know the absolute error after N instructions converges to 0 when ϵ goes to 0. This means, there is $\epsilon: \mathbb{R}_+$ for which the absolute error after executing N instructions is bounded by $\frac{1}{20}$. It is enough to show that four tests used in simulating \mathcal{A} are robust to $\frac{1}{20}$ absolute error. This is true since $\frac{3}{8} = 1 - \frac{5}{8} = \frac{5}{8} - \frac{1}{4}$ and $\frac{1}{12} = \frac{1}{6} - \frac{1}{12} = \frac{1}{4} - \frac{1}{6}$ are both strictly larger than $\frac{1}{20}$, and these are the error margins that each test allows before it fails to correctly simulate \mathcal{A} . Note that $\frac{1}{20} < 1$ therefore, x in Figure 5.1 is always in $[-2, 2]$. Finally, since we know the execution has at most N steps, and each step takes at most $\frac{1}{1-\epsilon}$ units of time, we set T to be $2N \geq \frac{N}{1-\epsilon}$. Note that after \mathcal{A} halts, it won't executes any more steps. This corresponds to a location in monotonic rectangular automaton with no outgoing transition. Therefore, although $2N$ is enough time to simulate more than N steps of \mathcal{A} , after the N^{th} step, \mathcal{A} is guaranteed to halt. So the extra time won't result in simulating more than N steps.

⁴Wlog., we use 20ϵ instead of 17ϵ , to make the arithmetic simpler.

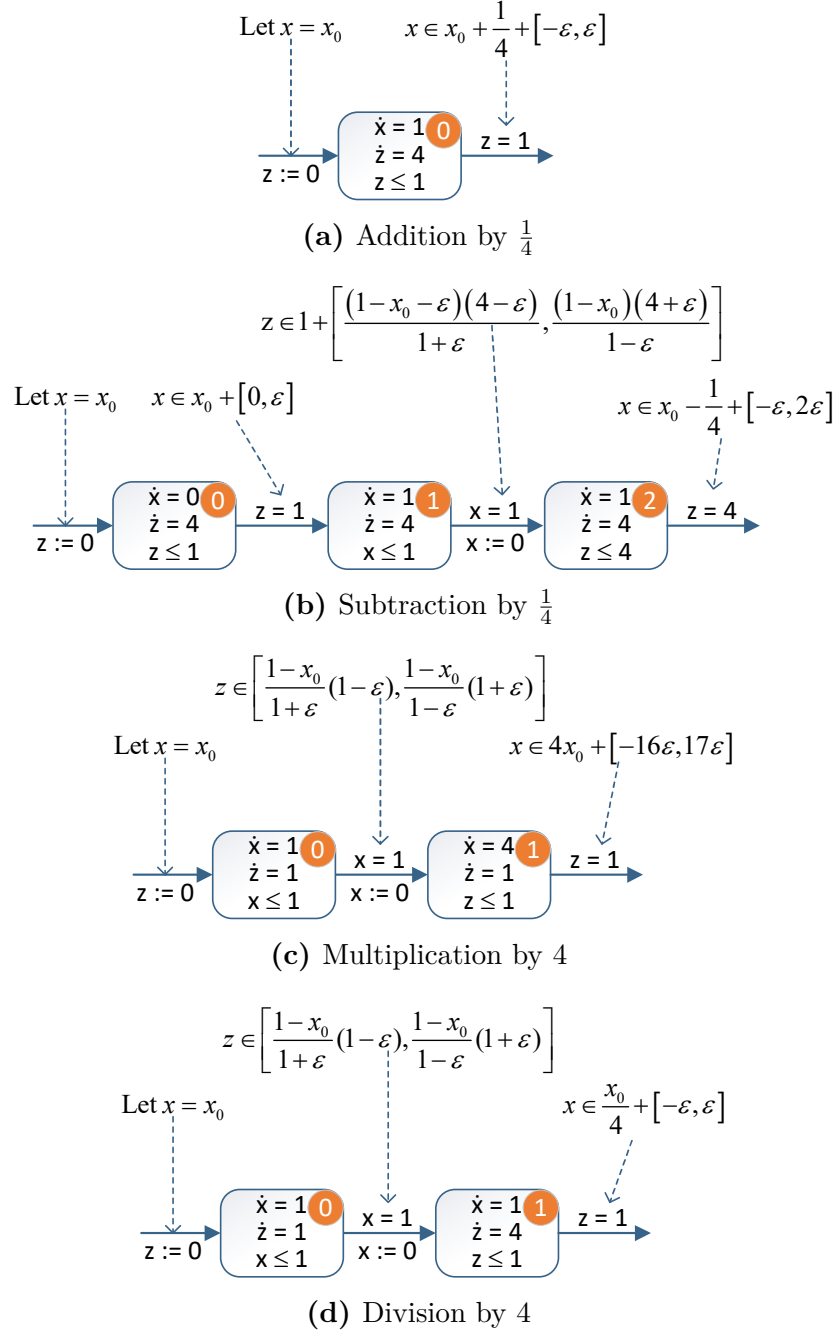


Figure 5.1: Gadgets for simulating Turing machine. In each gadget, initial value of x is x_0 and initial value of z is 0. After each discrete step, we show an over-approximation of possible values of the unknown variables. For example, in Figure 5.1b, after leaving location 0, value of x is in $[x_0, x_0 + \varepsilon]$ (value if z is guaranteed to be exactly 1 because of the guard). The total time each gadget takes to execute is at most $\frac{1}{1-\varepsilon}$. Also, since value of x is always less than 2, the absolute error of the actual result is within $[-16\varepsilon, 17\varepsilon]$ neighborhood of the expected result.

Theorem 39. Time bounded robust reachability problem for monotonic rectangular automata with only perturbed flows is in NEXPTIME-hard.

The next corollary follows immediately from Corollary 38 and Theorem 39.

Corollary 40. Time bounded robust reachability problem for monotonic rectangular automata with only perturbed flows is in NEXPTIME-complete.

5.2 Robust ω -Regular Model Checking

In this section, we present our results for the robust ω -regular model checking problem when both clocks and guards are perturbed and the robust ω -regular model checking problem when only clocks are perturbed. Recall that the robust ω -regular model checking problem when only guards are enlarged was solved in [86].

5.2.1 Robust ω -Regular Model Checking When Both Guards and Clocks are Perturbed

Our main result in this section is that robust ω -regular model checking problem when both guards and clocks are perturbed is equivalent to the robust ω -regular model checking problem when only guards are perturbed. This is formalized in the following theorem.

Theorem 41. For any timed automaton \mathcal{H} and B a subset of $E_{\mathcal{H}}$,

$$(\exists \epsilon, \delta : \mathbb{R}_+ \bullet \mathcal{H}_{\delta}^{\epsilon} \models B) \Leftrightarrow (\exists \delta' : \mathbb{R}_+ \bullet \mathcal{H}_{\delta'} \models B)$$

Note that the implication from left to right is trivial, since if $\mathcal{H}_{\delta}^{\epsilon} \models B$ then by assigning δ to δ' , we obtain $\mathcal{H}_{\delta'} \models B$ as well. The non-trivial part of the proof lies in showing the other direction. It requires the robustness under perturbed guards to be transferred to that under clock drifts and perturbed guards, which is facilitated by the following lemma.

Lemma 42. For any timed automaton \mathcal{H} , $\epsilon : (0, \frac{1}{4})$, $\gamma : (0, \frac{M}{2})$, and $\tau : \llbracket \mathcal{H}_{\gamma}^{\epsilon} \rrbracket_{\infty}^0$, there is $\tau' : \llbracket \mathcal{H}_{2M\epsilon+\gamma} \rrbracket_{\infty}^0$, such that τ and τ' have the exact same sequence of discrete and continuous transitions. Here, M is the largest constant appearing in the constraints of \mathcal{H} .

Proof. Fix $\epsilon: (0, \frac{1}{4})$, $\gamma: (0, \frac{M}{2})$, and $\tau: \llbracket \mathcal{H}_\gamma^\epsilon \rrbracket_\infty^0$. Let $\delta := 2\epsilon M + \gamma$. Obviously, the set of edges that are enabled initially in $\mathcal{H}_\gamma^\epsilon$ corresponds to a subset of the edges that are initially enabled in \mathcal{H}_δ . We show the same is true at any step in τ . For any variable $x: \mathbf{X}$ if valuation ν_x satisfies guard g at the time of taking edge e in $\mathcal{H}_\gamma^\epsilon$, there are two cases:

1. $g := x \triangleright c$ for some integer $c: \{0, \dots, M\}$ and $\triangleright: \{>, \geq\}$. Time taken by x to reach its current value since its last reset (or since the beginning of time if x has never been reset) is at least $\frac{c-\gamma}{1+\epsilon}$. We show that this time is big enough for the enlarged guard $x \triangleright c - \delta$ to be satisfied when x evolves without clock drifts.

$$\begin{aligned}
(M \triangleright c) & \Rightarrow \\
(2M + \gamma + 2\epsilon M \triangleright c) & \Rightarrow \\
(0 \triangleright -2\epsilon M + c\epsilon - \gamma\epsilon - 2\epsilon^2 M) & \Rightarrow \\
(c - \gamma \triangleright c - \gamma - 2\epsilon M + c\epsilon - \gamma\epsilon - 2\epsilon^2 M) & \Rightarrow \\
(c - \gamma \triangleright (1 + \epsilon)(c - \gamma - 2\epsilon M)) & \Rightarrow \\
\left(\frac{c - \gamma}{1 + \epsilon} \triangleright c - \gamma - 2\epsilon M = c - \delta \right) &
\end{aligned}$$

2. $g := x \triangleleft c$ for some integer $c: \{0, \dots, M\}$ and $\triangleleft: \{<, \leq\}$. Time taken by x to reach its current value, since its last reset (or since the beginning of time if x has never been reset) is at most $\frac{c+\gamma}{1-\epsilon}$. We show that this time is small enough for the enlarged guard $x \triangleleft c + \delta$ to be satisfied when there are no clock drifts. Again,

$$\begin{aligned}
(c \leq M) & \Rightarrow \\
\left(c + \frac{M}{2} + \frac{M}{2} \leq 2M \right) & \Rightarrow \\
(c + 2\epsilon M + \gamma \triangleleft 2M) & \Rightarrow \\
(0 \triangleleft 2\epsilon M - c\epsilon - 2\epsilon^2 M - \epsilon\gamma) & \Rightarrow \\
(c + \gamma \triangleleft c + \gamma + 2\epsilon M - c\epsilon - 2\epsilon^2 M - \epsilon\gamma) & \Rightarrow \\
(c + \gamma \triangleleft (1 - \epsilon)(c + 2\epsilon M + \gamma)) & \Rightarrow \\
\left(\frac{c + \gamma}{1 - \epsilon} \triangleleft c + 2\epsilon M + \gamma = c + \delta \right) &
\end{aligned}$$

Note that variables are reset at the exact same times in $\tau: \llbracket \mathcal{H}_\gamma^\epsilon \rrbracket_\infty^0$ and $\tau: \llbracket \mathcal{H}_\delta \rrbracket_\infty^0$. So whenever a variable is reset in $\tau: \llbracket \mathcal{H}_\gamma^\epsilon \rrbracket_\infty^0$ at time t , the same variable is reset in $\tau: \llbracket \mathcal{H}_\delta \rrbracket_\infty^0$ at time t . This makes the values of x in two

executions 0 at time t . □

The right to left implication of Theorem 41 follows directly from Lemma 42. If $\mathcal{H}_{\delta'} \models B$, then by choosing any $\epsilon: (0, \frac{1}{4})$ and $\delta: (0, \frac{M}{2})$ such that $2M\epsilon + \delta < \delta'$, we know any execution in $\llbracket \mathcal{H}_{\delta}^{\epsilon} \rrbracket_{\omega}^0$ has a corresponding execution in $\llbracket \mathcal{H}_{\delta'} \rrbracket_{\omega}^0$ with the same sequence of discrete transitions. Lemma 42 also implies $\text{limreach}_{\delta}^{\epsilon}(\mathcal{H})$ is subset of $\text{limreach}_{\delta}(\mathcal{H})$ and therefore they are equal even when the progress cycle assumption does not hold. This is formalized in the Corollary 43.

Corollary 43. For any bounded timed automaton \mathcal{H} , we have

$$\text{limreach}_{\delta}^{\epsilon}(\mathcal{H}) \subseteq \text{limreach}_{\delta}(\mathcal{H})$$

Proof. $\text{limreach}_{\delta}(\mathcal{H})$ is taken to be the intersection of the reach set of \mathcal{H}_{δ} for all δ . Since the guards are perturbed in \mathcal{H}_{δ} , we can (without loss of generality) assume that the intervals are closed in \mathcal{H}_{δ} when defining $\text{limreach}_{\delta}(\mathcal{H})$ [85]. Thus, $\text{reach}(\mathcal{H}_{\delta})$ is a closed set, for any $\delta: \mathbb{R}_+$. Let $s: \text{limreach}_{\delta}^{\epsilon}(\mathcal{H})$ be an arbitrary state. For any $\delta: \mathbb{R}_+$, we will show $s \in \text{reach}(\mathcal{H}_{\delta})$, thus establishing the claim. Since $\text{reach}(\mathcal{H}_{\delta})$ is closed, we can establish this by showing that $d_{\infty}(s, \text{reach}(\mathcal{H}_{\delta}))$ is arbitrarily small, or more precisely, for any $\kappa: \mathbb{R}_+$, $d_{\infty}(s, \text{reach}(\mathcal{H}_{\delta})) < \kappa$.

Let us fix $\kappa: \mathbb{R}_+$. Pick $\epsilon: (0, \frac{1}{4})$ and $\gamma: (0, \frac{M}{2})$ such that $2M\epsilon + \gamma < \delta$ and $2M\epsilon < \kappa$. Since $s \in \text{reach}(\mathcal{H}_{\gamma}^{\epsilon})$ we know there is an execution $\tau: \llbracket \mathcal{H}_{\gamma}^{\epsilon} \rrbracket_{*}^0$ such that $\text{last}(\tau) = s$. By Lemma 42 we know there is an execution $\tau': \llbracket \mathcal{H}_{2M\epsilon+\gamma} \rrbracket_{*}^0 \subseteq \llbracket \mathcal{H}_{\delta} \rrbracket_{*}^0$ with the same sequence of discrete and continuous transitions as τ . Since \mathcal{H} is bounded and $\epsilon < \frac{1}{2}$, every variable is reset at least once every $2M$ units of time. Since the clocks are drifted by at most ϵ in $\mathcal{H}_{\gamma}^{\epsilon}$ and variables are reset at the exact same times in τ and τ' , $d_{\infty}(s, \text{last}(\tau')) \leq 2M\epsilon < \kappa$. Therefore, $d_{\infty}(s, \text{reach}(\mathcal{H}_{\delta})) < \kappa$. □

To obtain an algorithm for robust ω -regular model checking problem when both guards and clocks are perturbed, we resort to the results in [86] for robust ω -regular model checking problem when only guards are perturbed. The algorithm in [86] provides a computable value δ_0 , such that the $\mathcal{H}_{\delta_0} \models B$ iff there is $\delta: \mathbb{R}_+$ such that $\mathcal{H}_{\delta} \models B$. We recall this result from [86].

Theorem 44 (Based on Lemma 11 and Theorem 3 in [86]). Let \mathcal{H} be a timed automaton, W be its number of regions, and B be a subset of $E_{\mathcal{H}}$. Then

1. $\forall \delta : \mathbb{R}_+, \tau : \llbracket \mathcal{H}_{\delta_0} \rrbracket_{\omega}^0 \bullet \exists \tau' : \llbracket \mathcal{H}_{\delta} \rrbracket_{\omega}^0 \bullet \inf(\tau) = \inf(\tau')$
2. $\exists \delta : \mathbb{R}_+ \bullet \mathcal{H}_{\delta} \models B \Leftrightarrow \mathcal{H}_{\delta_0} \models B$

where $\delta_0 := \frac{1}{2}(8|X|^2(W+1))^{-1}$ if \mathcal{H} satisfies the progress cycle assumption, otherwise $\delta_0 := \frac{1}{2}(5(W+1)|X|^3(2|Q|(|X|!)4^{|X|} + 4)^2)^{-1}$.

Proof. The second result follows immediately from the first one. As far as the first result is concerned, the only difference when compared to the observations in [86], is that here $\inf(\cdot)$ returns the set of edges, as opposed to the set of locations, that are visited infinitely often. This is not a problem since the proof in [86] essentially obtains τ' from τ by 1. repeating some subexecutions of τ a finite number of times (duration of continuous transitions may change during this step), and 2. repeating the previous step for a finite number of times. \square

In order to solve robust ω -regular model checking when both guards and clocks are perturbed, our algorithm first checks $\mathcal{H}_{\delta_0} \models B$ where δ_0 is introduced in Theorem 44. If $\mathcal{H}_{\delta_0} \not\models B$ then clearly \mathcal{H} does *not* robustly satisfy B when both guards and clocks are perturbed. On the other hand, if $\mathcal{H}_{\delta_0} \models B$ then using Lemma 42 we know $\mathcal{H}_{\delta_1}^{\epsilon_1} \models B$ for any $\delta_1, \epsilon_1 : \mathbb{R}_+$ that satisfy $2M\epsilon_1 + \delta_1 < \delta_0$.

Theorem 45. Let \mathcal{H} be a timed automaton and B be a subset of $E_{\mathcal{H}}$. Let δ_0 be as defined in Theorem 44. Then, for any, ϵ_1, δ_1 such that $2M\epsilon_1 + \delta_1 \leq \delta_0$, the following holds:

$$(\exists \epsilon, \delta : \mathbb{R}_+ \bullet \mathcal{H}_{\delta}^{\epsilon} \models B) \Leftrightarrow (\mathcal{H}_{\delta_0} \models B) \Leftrightarrow (\mathcal{H}_{\delta_1}^{\epsilon_1} \models B)$$

Since the robust ω -regular model checking problem when only guards are perturbed is PSPACE-complete, from Theorem 45, we obtain the following:

Corollary 46. The robust ω -regular model checking problem when both guards and clocks are perturbed is PSPACE-complete.

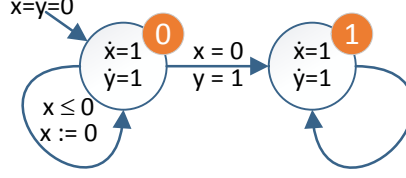


Figure 5.2: $\exists \epsilon : \mathbb{R}_+ \bullet \mathcal{H}^\epsilon \models B$ does not imply $\exists \delta : \mathbb{R}_+ \bullet \mathcal{H}_\delta \models B$

While robust ω -regular model checking problem when both guards and clocks are perturbed and when only guards are perturbed are equivalent (Theorem 41), the same is surprisingly not true for the case when only clocks drift. This is demonstrated by our next example.

Example 47. In this example, we show that

$$(\exists \epsilon : \mathbb{R}_+ \bullet \mathcal{H}^\epsilon \models B) \not\Rightarrow (\exists \delta : \mathbb{R}_+ \bullet \mathcal{H}_\delta \models B)$$

Consider the timed automaton \mathcal{H} in Figure 5.2. It starts at location 0 with $x = y = 0$ and both the variables evolve at rate 1. Note that e_1 , the self loop on location 0, can only be taken when $x = 0$, that is, it is feasible only at time 0. Hence, the edge e_2 from location 0 to location 1 can never be taken, since once time passes both x and y will have non-zero values and the value of x can never be reset.

Taking $B := \{e_1\}$, consider the ω -regular property B . Note that the only infinite execution τ of the timed automaton is the one which traverses e_1 repeatedly without time elapsing. This satisfies the condition B . Moreover, even if we allow clock drifts, we will still have only one infinite execution, namely, τ . Hence, $\mathcal{H}^\epsilon \models B$ (for any $\epsilon : [0, 1)$).

However, we argue that for any δ , $\mathcal{H}_\delta \not\models B$, since, there will be infinite trajectories that visit location 1 and execute e_3 , the self loop on location 1, repeatedly. To see this, note that a δ enlargement of the guard $x \leq 0$ will lead to $x \leq \delta$. Hence, by repeatedly taking e_1 every δ units of time, the value $x = 0$ and $y \in [1 - \delta, 1 + \delta]$ can be reached. Hence, the edge to location 1 can be taken.

Observe that the same example works if we bound guards and invariants. Furthermore, this example also shows that $\text{limreach}^\epsilon(\mathcal{H}) \neq \text{limreach}_\delta(\mathcal{H})$, a fact we mentioned in the introduction.

5.2.2 Robust ω -Regular Model Checking When Only Clocks are Perturbed

In Example 47, we showed that even if there exists an $\epsilon : \mathbb{R}_+$ such that $\mathcal{H}^\epsilon \models B$, there may not exist a $\delta : \mathbb{R}_+$ such that $\mathcal{H}_\delta \models B$. However, we show that the implication holds when we consider a weaker notion of guard perturbations.

Theorem 48. For any timed automaton \mathcal{H} and B a subset of \mathbf{E} ,

$$(\exists \epsilon : \mathbb{R}_+ \bullet \mathcal{H}^\epsilon \models B) \Leftrightarrow (\exists \delta : \mathbb{R}_+ \bullet \mathcal{H}_{+\delta} \models B)$$

The crux of the proof of Theorem 48 lies in the following lemma which establishes the connection between the trace language of automata under clock drifts and the trace language of automata when positive guard constants are perturbed.

Lemma 49. For a timed automaton \mathcal{H} with maximum constant M and $\epsilon : (0, \frac{1}{2})$ the following holds:

$$\text{trace}(\llbracket \mathcal{H}_{+\frac{\epsilon}{2}} \rrbracket_\infty^0) \subseteq \text{trace}(\llbracket \mathcal{H}^\epsilon \rrbracket_\infty^0) \subseteq \text{trace}(\llbracket \mathcal{H}_{+2M\epsilon} \rrbracket_\infty^0)$$

Proof. \mathcal{H}^ϵ is an initialized rectangular automaton. In [2], Henzinger *et al.* describe a transformation from 1. an initialized rectangular automaton \mathcal{H}^ϵ into an initialized monotonic rectangular automaton \mathcal{H}' that is trace equivalent with \mathcal{H}^ϵ , and 2. \mathcal{H}' to a timed automaton \mathcal{H}'' which is bisimilar to \mathcal{H}' .

Initialized monotonic rectangular automaton \mathcal{H}' is obtained from \mathcal{H} by replacing every variable $x : \mathbf{X}_\mathcal{H}$ by two variables x_l and x_u . Initially all variables in \mathcal{H}' are zero. \mathcal{H} and \mathcal{H}' have the same set of (initial) locations and the invariant in every location in \mathcal{H}' is the set of all valuations. For any location $q : \mathbf{Q}_\mathcal{H}$ and variable $x : \mathbf{X}_\mathcal{H}$, $\mathbf{F}_{\mathcal{H}'}(q, x_l) = 1 - \epsilon$ and $\mathbf{F}_{\mathcal{H}'}(q, x_u) = 1 + \epsilon$ (recall that $\mathbf{F}(q, x)$ is rate at which variable x changes in location q). There is a bijection between edges of \mathcal{H} and \mathcal{H}' . For any edge $e : \mathbf{E}_\mathcal{H}$ there is an edge $e' \in \mathbf{E}_{\mathcal{H}'}$ such that 1. e' has the same source and destination as e , 2. For any $x : \mathbf{X}_\mathcal{H}$ we have $x \in \mathbf{R}_\mathcal{H}e \Leftrightarrow x_l, x_u \in \mathbf{R}_{\mathcal{H}'}e'$, 3. For any $x : \mathbf{X}_\mathcal{H}$ and constant $c : \mathbb{R}$ we have $(x \leq c) \in \mathbf{G}_\mathcal{H}e \Leftrightarrow (x_l \leq c) \in \mathbf{G}_{\mathcal{H}'}e'$ and $(x \geq c) \in \mathbf{G}_\mathcal{H}e \Leftrightarrow (x_u \geq c) \in \mathbf{G}_{\mathcal{H}'}e'$.

Timed automaton \mathcal{H}'' is obtained from \mathcal{H}' by 1. setting $\mathbf{F}_{\mathcal{H}''}(q, x_l) = \mathbf{F}_{\mathcal{H}'}(q, x_u) = 1$ for all $q : \mathbf{Q}_\mathcal{H}$ and $x : \mathbf{X}_\mathcal{H}$. 2. every guard $x_l \leq c$ is replaced by $x_l \leq \frac{c}{1-\epsilon}$, and 3. every guard $x_u \geq c$ is replaced by $x_u \geq \frac{c}{1+\epsilon}$. Note that x_l and

x_u are both reset to zero at the exact same times, and they are both initially zero. Therefore, one can merge these two variables into one and combine their guards together. This is possible since \mathcal{H} is a timed automaton.

If $c = 0$ then $\frac{c}{1-\epsilon} = \frac{c}{1+\epsilon} = 0$. This corresponds to not changing zero guards. Next we show for $c \geq 1$ the following are true: 1. $c + \epsilon \leq \frac{c}{1-\epsilon} \leq c + 2M\epsilon$, and 2. $c - M\epsilon \leq \frac{c}{1+\epsilon} \leq c - \frac{\epsilon}{2}$.

$$\begin{aligned}
\left(\frac{c}{1-\epsilon} \leq c + 2M\epsilon\right) &\Leftrightarrow (c \leq c - c\epsilon + 2M\epsilon - 2M\epsilon^2) \Leftrightarrow (c + 2M\epsilon \leq 2M) \\
\left(c + \epsilon \leq \frac{c}{1-\epsilon}\right) &\Leftrightarrow (c - c\epsilon + \epsilon - \epsilon^2 \leq c) \Leftrightarrow (1 - \epsilon \leq c) \\
\left(\frac{c}{1+\epsilon} \leq c - \frac{\epsilon}{2}\right) &\Leftrightarrow \left(c \leq c + c\epsilon - \frac{\epsilon}{2} - \frac{\epsilon^2}{2}\right) \Leftrightarrow \left(\frac{1}{2} + \frac{\epsilon}{2} \leq c\right) \\
\left(c - M\epsilon \leq \frac{c}{1+\epsilon}\right) &\Leftrightarrow (c + c\epsilon - M\epsilon - M\epsilon^2 \leq c) \Leftrightarrow (c - M\epsilon \leq M)
\end{aligned}$$

This means that if we enlarge guards by $2M\epsilon$, any time a guard is enabled in \mathcal{H}'' the same guard is enabled in $\mathcal{H}_{+2M\epsilon}$. Similarly, if we enlarge guards by $\frac{\epsilon}{2}$, any time a guard is enabled in $\mathcal{H}_{+\frac{\epsilon}{2}}$ the same guard is enabled in \mathcal{H}'' . The proof is complete once we remember that \mathcal{H}^ϵ and \mathcal{H}'' are trace equivalent. \square

Next, we present an algorithm for robust ω -regular model checking problem when only positive guards are perturbed. The main technical observation is that there exists a computable δ_1 such that for all positive guard perturbations $\delta \in (0, \delta_1)$, the set of edges that are visited infinitely often along the executions of $\mathcal{H}_{+\delta_1}$ are contained in that of $\mathcal{H}_{+\delta}$. Therefore, if $\mathcal{H}_{+\delta} \models B$ for some δ , then $\mathcal{H}_{+\delta_1} \models B$ (the other implication is trivial). These observations are captured in Lemma 50 and Theorem 51. We first need a few definitions.

For any timed automaton \mathcal{H} and any two edges $e_1, e_2 : \mathbf{E}$, let $\text{succ}(e_1, e_2)$ be a predicate that returns true iff 1. $\text{De}_1 = \text{Se}_2$, and 2. $\forall x : \text{Re}_1 \bullet (x = 0) \wedge \text{Ge}_2 \not\models \perp$. The second condition means that if x is reset by e_1 then predicate $x = 0$ is consistent with the guard of e_2 . Intuitively, $\text{succ}(e_1, e_2)$ is true iff e_1 and e_2 can be merged. When this is the case, we define $e := \text{merge}(e_1, e_2)$ to be the edge $(\text{Se}_1, \text{De}_2, \text{Ge}_1 \wedge g, \text{Re}_1 \cup \text{Re}_2, \text{Le}_1 \cup \text{Le}_2 \cup \{e_1, e_2\})$ where for any guard $(x \bowtie c) : \text{Ge}_2$ if $x \notin \text{Re}_1$ then $(x \bowtie c) \in g$, otherwise, g does not constrain x . It is easy to see that if $s_1 \xrightarrow{e_1} s_2 \xrightarrow{0} s_3 \xrightarrow{e_2} s_4$, for some $s_1, s_2, s_3, s_4 : \mathbf{S}_{\llbracket \mathcal{H} \rrbracket}$ then $s_1 \xrightarrow{e} s_4$. Let \mathbf{E}^s be the smallest set that contains \mathbf{E} and for any two edges $e_1, e_2 : \mathbf{E}^s$ if $\text{succ}(e_1, e_2)$ then $\text{merge}(e_1, e_2) \in \mathbf{E}^s$. Let \mathcal{H}^s be same as \mathcal{H} except

its set of edges is replaced by \mathbf{E}^s and q is an initial location in \mathcal{H}^s iff $q \in \mathbf{Q}^{\text{init}}_{\mathcal{H}}$ or $(q', 0^{\mathbf{x}}) \xrightarrow{e} (q, 0^{\mathbf{x}})$ for some $q' : \mathbf{Q}^{\text{init}}_{\mathcal{H}}$ and $e : s \rightarrow \mathbf{E}$, where $0^{\mathbf{x}}$ is a function that maps every variable in \mathbf{x} to 0. Intuitively, q is an initial location in \mathcal{H}^s iff q is an initial location of \mathcal{H} or it can be reached using an execution of duration 0. Note that $|\mathbf{E}^s| < \infty$ therefore \mathcal{H}^s is a well defined timed automaton. Finally, by construction, for any two states $s, s' : \mathbf{S}_{\llbracket \mathcal{H} \rrbracket} = \mathbf{S}_{\llbracket \mathcal{H}^s \rrbracket}$ and $e : \mathbf{E}^s$ if $s \xrightarrow{e} s'$ and e is a merge of $e_1, \dots, e_n : \mathbf{E}_{\mathcal{H}}$ in the given order (we don't need merge to be associative) then $s_0 \xrightarrow{e_1} s'_1 \xrightarrow{0} s_1 \xrightarrow{e_2} \dots \xrightarrow{0} s'_n \xrightarrow{e_n} s_n$ for some $s_0, \dots, s_n, s'_1, \dots, s'_n : \mathbf{S}_{\llbracket \mathcal{H} \rrbracket}$ such that $s_0 = s$ and $s_n = s'$. Furthermore, $\mathbf{Le} = \{e_1, \dots, e_n\}$.

Lemma 50. Let \mathcal{H} be a timed automaton and $\delta_1 := \frac{\delta_0}{24}$, where δ_0 is defined as in Theorem 44. For any $\delta : (0, \delta_1)$ and execution $\tau : \llbracket \mathcal{H}_{+\delta_1} \rrbracket_{\omega}^0$ there is an execution $\tau' : \llbracket \mathcal{H}_{+\delta} \rrbracket_{\omega}^0$ such that $\text{inf}(\tau) = \text{inf}(\tau')$.

Proof. If $\text{duration}(\tau) = 0$ the theorem is trivially true. Therefore, for the rest of the proof assume $\text{duration}(\tau) > 0$.

1. Suppose τ has infinitely many non-zero time transitions. We prove this part for $\delta_1 := \delta_0$ (which is stronger than $\delta_1 := \frac{\delta_0}{24}$). After any non-zero time transition, value of no variable is zero. Therefore, none of the guards of the form $x \leq 0$ are satisfied. Construct an execution $u : \llbracket \mathcal{H}_{+\delta_1}^s \rrbracket_{\omega}^0$ from τ by removing all zero time transitions and merging edges that their in-between-time-transitions have been removed. Also, if u does not start with non-zero time transition, remove e_0 the first discrete transition of it. After this we know that u starts with a non-zero time transition and $x \leq 0$ is never used in its guards. Theorem 44 only uses edges of u (i.e. existence of other edges is neither assumed nor proved) and guards of u are the same in $\mathcal{H}_{+\delta_1}^s$ and $\mathcal{H}_{\delta_1}^s$. Therefore, by Theorem 44 there is an execution $u' : \llbracket \mathcal{H}_{+\delta}^s \rrbracket_{\omega}^0$ such that $\text{inf}(u') = \text{inf}(u)$. If we replace any edge e in u that belongs to $\mathbf{E}^s \setminus \mathbf{E}$ with some sequence of edges from \mathbf{Le} and do the same for e_0 (if it was initially removed), we get an execution $\tau'' : \llbracket \mathcal{H}_{+\delta_1} \rrbracket_{\omega}^0$. If we do the same to u' , we get an execution $\tau' : \llbracket \mathcal{H}_{+\delta} \rrbracket_{\omega}^0$. Furthermore, we know $\text{inf}(\tau') = \text{inf}(\tau'')$. This part is complete once we notice $\text{inf}(\tau) = \text{inf}(\tau'')$ since edges in \mathbf{E}^s keep track of the set of edges that they visit internally.

2. Next suppose $\tau := \tau^1 \frown \tau^2$ for some $\tau^1 : \llbracket \mathcal{H}_{+\delta_1} \rrbracket_*^0$ and $\tau^2 : \llbracket \mathcal{H}_{+\delta_1} \rrbracket_\omega$ such that $\text{duration}(\tau^2) = 0$. Let $Y \subseteq \mathbf{X}$ be the possibly empty set of variables such that for any $x : Y$ the guard $x \leq 0$ occurs infinitely often in τ . Let $\tau := \tau^3 \frown \tau^4$ for some $\tau^3 : \llbracket \mathcal{H}_{+\delta_1} \rrbracket_*^0$ and $\tau^4 : \llbracket \mathcal{H}_{+\delta_1} \rrbracket_\omega$ such that $\text{duration}(\tau^4) = 0$, all variables in Y are always zero in τ^4 , and $x \leq 0$ appears in guards of τ^4 only for $x : Y$. For any variable $x : \mathbf{X}$, let $\tau_x^4 : \llbracket \mathcal{H}_{+\delta_1} \rrbracket_\infty$ be the longest prefix of τ^4 that either never resets x or resets x only in its last step. Also, let g_x be the conjunction of all guards on x in τ_x^4 that are not $x \leq 0$. Furthermore, let $g := \bigwedge_{x : \mathbf{X}} g_x$, and let g_δ be g enlarged by δ . Note that g_{δ_1} is satisfied by $\text{last}(\tau^3)$. Let q be the last location visited in τ^3 . Add a new location q' to $\mathbf{Q}_\mathcal{H}$ and two new edges e and e' to $\mathbf{E}_\mathcal{H}$. e is from q to q' with guard $g \wedge \bigwedge_{x : Y} x \leq 0$ that resets no variable, and e' is a self-loop on q' with guard g that also resets no variable. Call the new timed automaton \mathcal{H}' .

Let $u := u^1 \frown u^2$ for some $u^1 : \llbracket \mathcal{H}'_{+2\delta_1} \rrbracket_*^0$. Also let $u^2 : \llbracket \mathcal{H}'_{+2\delta_1} \rrbracket_\omega$ such that $u^1 := \text{first}(\tau^3) \xrightarrow{\tau^3} \text{last}(\tau^3) \xrightarrow{e} s$ and $u^2 := s_n \xrightarrow{t_n} s'_n \xrightarrow{e'} s_{n+1}$ for some $s, s_0, s'_0, s_1, s'_1, s_2, s'_2, \dots : \mathbf{S}_{\llbracket \mathcal{H}' \rrbracket}$ where $s_0 = s$ and $t_n := \frac{\delta_1}{2^{n+1}}$ for all $n : \mathbb{N}$. Since $\text{duration}(u^2) = \delta_1$ we know $g_{2\delta_1}$ is always satisfied in u^2 and therefore u is a valid execution.

Let δ'_0 be the maximum enlargement Theorem 44 specifies for \mathcal{H}' . Since \mathcal{H}' has only one location more than \mathcal{H} and number of edges is not used in Theorem 44, we know $2\delta_1 \leq \delta'_0$. So we can use the previous case and find $u' : \llbracket \mathcal{H}'_{+\delta} \rrbracket_\omega^0$ such that $\inf(u) = \inf(u') = \{e'\}$. By construction of \mathcal{H}' , we know u' can be written as $u'^1 \frown u'^2$ for some $u'^1 : \llbracket \mathcal{H}'_{+\delta} \rrbracket_*^0$ and $u'^2 : \llbracket \mathcal{H}'_{+\delta} \rrbracket_\omega$ such that the last edge in u'^1 is e , e is used only once in u'^1 , and the only edge in u'^2 is e' .

Inside u' , right before and after taking e , g_δ is satisfied and all variables in Y are zero. Let $\tau' := \tau'^1 \frown \tau'^2$, where τ'^1 is obtained from u'^1 by removing e and the time transition after that, and τ'^2 is obtained from u'^2 by using the same trace and setting all time transitions equal to zero. We show that $\tau' \in \llbracket \mathcal{H}_{+\delta} \rrbracket_\omega^0$.

We know at the end of τ'^1 and everywhere in τ'^2 all variables in Y are zero which is the same in τ^4 . For any variable $x \notin Y$ we know the value of x satisfies g_δ at the end of τ'^1 and this value does not change in τ'^2 until x gets reset to 0. During this time x always satisfies g_δ and after

it gets reset to 0 it remains 0 both in τ'^2 and τ^4 .

□

Theorem 51. For any timed automaton \mathcal{H} and B a subset of E ,

$$(\exists \delta : \mathbb{R}_+ \bullet \mathcal{H}_{+\delta} \models B) \Leftrightarrow (\mathcal{H}_{+\delta_1} \models B)$$

where δ_1 is as defined in Lemma 50.

The above observations together give us the following result.

Theorem 52. Let \mathcal{H} be a timed automaton with maximum constant M and let B be a subset of E . Let δ_1 be as defined in Lemma 50, and let $\epsilon_1 := \frac{\delta_1}{2M}$. Then:

$$(\exists \epsilon : \mathbb{R}_+ \bullet \mathcal{H}^\epsilon \models B) \Leftrightarrow (\mathcal{H}_{+\delta_1} \models B) \Leftrightarrow (\mathcal{H}^{\epsilon_1} \models B)$$

Proof. Using Theorem 48 and Lemma 50 we only need to show $\mathcal{H}^{\epsilon_1} \models B$ is implied by $\exists \epsilon : \mathbb{R}_+ \bullet \mathcal{H}^\epsilon \models B$. If $\epsilon \geq \epsilon_1$ then we are done. Otherwise, *wlog.*, we can assume $\epsilon < \frac{1}{2}$ and using Lemma 49 we know for any $\tau : \llbracket \mathcal{H}^{\epsilon_1} \rrbracket_\omega^0$ if τ visits any element of B infinitely often then there is $\tau' : \llbracket \mathcal{H}_{+2M\epsilon_1} \rrbracket_\omega^0$ that visits the same set infinitely often. Next, using $2M\epsilon_1 \leq \delta_1$ and Theorem 48, there is $\tau'' : \llbracket \mathcal{H}_{+\frac{\epsilon}{2}} \rrbracket_\omega^0$ that visits the same set of edges infinitely often. Finally, using Lemma 49 we know there is $\tau''' : \llbracket \mathcal{H}^\epsilon \rrbracket_\omega^0$ that visits the same element in B infinitely often. □

Theorem 52 gives a simple algorithm to solve robust ω -regular model checking problem when only clocks drift — compute δ_1 and check that $\mathcal{H}_{+\delta_1} \models B$. Since $\mathcal{H}_{+\delta_1}$ is a timed automaton, δ_1 is only exponentially small, and ω -regular model checking problem for timed automaton can be solved in PSPACE, we have a simple upper bound on the complexity. We show that in fact this complexity bound is tight.

Corollary 53. The robust ω -regular model checking problem when only clocks drift is PSPACE-complete.

Proof. PSPACE-completeness when guards are enlarged by δ_0 has been established in [86]. Bouyer *et al.* proved for any poly-space bounded Turing machine \mathcal{A} there is a timed automaton \mathcal{H} and property B such that \mathcal{A} accepts the zero input iff $\mathcal{H}_{\delta_0} \models B$ and iff $\mathcal{H} \models B$. With the help of Lemma 42

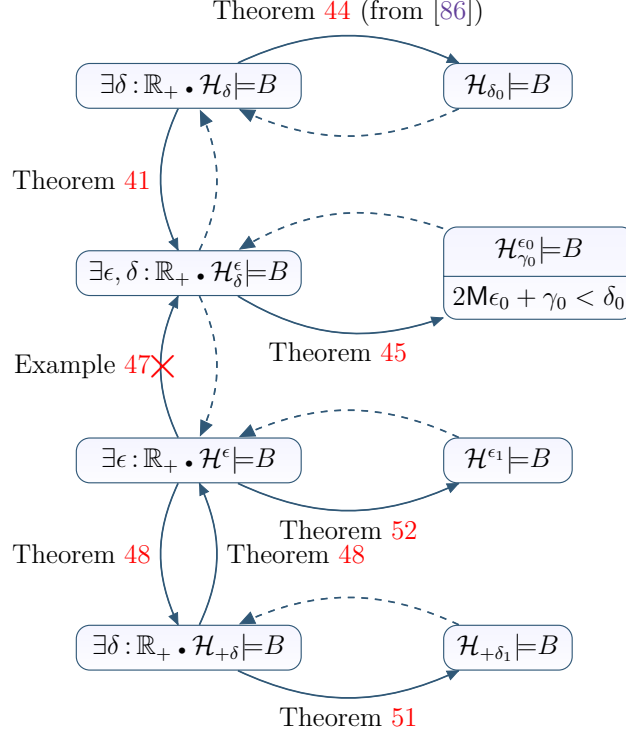


Figure 5.3: Overview of the results in Section 5.2. δ_0 is defined in Theorem 44, $\delta_1 := \frac{\delta_0}{24}$, $\epsilon_1 := \frac{\delta_1}{2M}$, and M is the maximum constant appeared in \mathcal{H} . Arrows are implications. Implications of dashed arrows are obvious. All problems are in PSPACE-complete.

and using $\epsilon_0 := \frac{\delta_0}{2M}$ we have $(\mathcal{H} \models B) \Rightarrow (\mathcal{H}_{\delta_0} \models B) \Rightarrow (\mathcal{H}^{\epsilon_0} \models B) \Rightarrow (\mathcal{H} \models B)$. Therefore, \mathcal{A} accepts the zero input iff $\mathcal{H}^{\epsilon_0} \models B$. \square

5.3 Computing Limit Reachable Sets under Clock Drifts

In this section, we will prove that $\text{limreach}^\epsilon(\mathcal{H})$ is computable in polynomial space for a *bounded* timed automaton \mathcal{H} . Recall that, a timed automaton \mathcal{H} is bounded if the invariants of \mathcal{H} are bounded by M , the maximum constant appearing in the constraints of \mathcal{H} . Restricting our attention to bounded timed automata is not limiting. It is well known [87, 88], that for any timed automaton \mathcal{H} , one can construct a bounded timed automaton \mathcal{H}' that is weakly bisimilar to \mathcal{H} . In fact, the construction also works when \mathcal{H} is perturbed (See Section 5.3.1 for formal proof). Thus, given that we show how

Algorithm 3 Computing the limit reachable set of a bounded timed automaton

Input: Bounded timed automaton \mathcal{H}

Output: $\text{limreach}^\epsilon(\mathcal{H})$

1. $\mathcal{G} \leftarrow$ the region graph of \mathcal{H}
 2. $W \leftarrow$ number of nodes in \mathcal{G}
 3. $\mathcal{C} \leftarrow$ the set of progress cycles in \mathcal{G} of length $\leq W + 2W|\mathbf{x}|$
 4. $J^* \leftarrow \bigcup_s : s^{\text{init}} \text{ reach}(\mathcal{G}, [s])$
 5. **while** $\exists p : \mathcal{C} \bullet [p_0] \not\subseteq J^* \wedge J^* \cap [p_0] \neq \emptyset$ **do**
 6. $J^* \leftarrow J^* \cup [p_0]$
 7. $J^* \leftarrow \text{reach}(\mathcal{G}, J^*)$
 8. **end while**
 9. **return** J^*
-

to compute $\text{limreach}^\epsilon(\mathcal{H})$ for a bounded automaton \mathcal{H} , we can compute the set of regions containing states that belong to $\text{limreach}^\epsilon(\mathcal{H}')$ for any (not necessarily bounded) timed automaton \mathcal{H}' .

The algorithm to compute $\text{limreach}^\epsilon(\mathcal{H})$ is very similar to the one for timed automata under the progress cycle assumption, and is shown as Algorithm 3. Lines 1 to 3 compute the set \mathcal{C} of all progress cycles whose length are bounded by $W + 2W|\mathbf{x}|$, where W is the number of regions of \mathcal{H} . The rest of the algorithm is similar to the approach in [84, 85]; the algorithm first computes the reachable regions assuming there is no perturbation (Line 4). It then iteratively adds closure of initial regions of progress cycles that are not in the current set of reachable regions but have non-empty intersection with it, and recomputes the reachable regions, until a fixpoint is reached. At that point the set J^* is the set of limit reachable states of \mathcal{H} (Line 9).

The main observation in this section is that J^* is indeed $\text{limreach}^\epsilon(\mathcal{H})$, which is stated next. Later in the section we argue that Algorithm 3 can in fact, be implemented in such a way that it uses only polynomial space.

Theorem 54. For a timed automaton \mathcal{H} , let J^* be the set returned on Line 9 in Algorithm 3. We have:

$$\text{limreach}^\epsilon(\mathcal{H}) = J^*$$

The proof of $J^* \subseteq \text{limreach}^\epsilon(\mathcal{H})$ follows from the observations in [85]⁵. To show that $\text{limreach}^\epsilon(\mathcal{H}) \subseteq J^*$, from Algorithm 3, we observe that J^* is a (topologically) closed set. If we show that $\text{d}_\infty(\text{limreach}^\epsilon(\mathcal{H}), J^*) = 0$, then the result follows immediately. This is the crux of the proof. We state this observation in a slightly stronger form below as Lemma 58. Before presenting this lemma and its proof, we recall 2 observations from [85] that we will use. The first observation is that if the distance between the closure of two regions is small, then they have a non-empty intersection.

Lemma 55 (Lemma 16 in [85]). For any timed automaton \mathcal{H} and for any two regions r_1 and r_2 , if $\text{d}_\infty([r_1], [r_2]) < \frac{1}{|\mathbf{x}|}$ then $[r_1] \cap [r_2] \neq \emptyset$.

Next, we recall a result from [85] that shows that for any distance bound α and number of steps k , all trajectories of certain perturbed timed automata $\mathcal{H}_\delta^\epsilon$ are close to some trajectory of \mathcal{H} .

Theorem 56 (Theorem 44 in [85]). Let \mathcal{H} be a bounded timed automaton. For any distance $\alpha : (0, 1)$ and number of steps $k : \mathbb{N}$, there are two numbers $D(\alpha, k), E(\alpha, k) : \mathbb{R}_+$ such that for any $\epsilon : [0, E(\alpha, k)]$, $\delta : [0, D(\alpha, k)]$, and a trajectory $\tau' : \llbracket \mathcal{H}_\delta^\epsilon \rrbracket_*$ such that $|\tau'| \leq k$ there is a trajectory $\tau : \llbracket \mathcal{H} \rrbracket_*$ with the following properties:

1. $\text{first}(\tau) \in [\text{first}(\tau')]$
2. $\text{trace}(\tau) = \text{trace}(\tau')$,
3. τ is *close* to τ' in the following sense: $\forall i : \{0, \dots, |\tau|\} \bullet \text{d}_\infty(\tau_{2i}, \tau'_{2i}) < \alpha$.

Finally, we recall a result about monotonic rectangular automata from Section 5.1 that we apply to clock drifted timed automata. The result states that corresponding to any trajectory of duration T between two states in $\mathcal{S}_{\mathcal{H}^\epsilon}$, there is a trajectory of the same duration between the same states such that number of steps is bounded by a function of T and \mathcal{H} . We state this observation for \mathcal{H}^ϵ .

⁵The algorithm in [85] only considers simple cycles, i.e., cycles whose length is bounded by W . However, the proof of $J^* \subseteq \text{limreach}^\epsilon(\mathcal{H})$ in [85] does not rely on the assumption, and so even though our algorithm may consider non-simple cycles in its analysis, this does not result in any additional reachable states.

Corollary 57 (Based on Theorem 36). Let \mathcal{H} be a timed automaton with M as the maximum constant appearing in its constraints, $\epsilon : [0, 1]$ be a perturbation, and $T : \mathbb{R}_{\geq 0}$ be a time bound. Then for any $\tau : \llbracket \mathcal{H}^\epsilon \rrbracket_*$ there is $\tau' : \llbracket \mathcal{H}^\epsilon \rrbracket_*$ with the following properties:

1. $\text{first}(\tau') = \text{first}(\tau)$,
2. $\text{last}(\tau') = \text{last}(\tau)$,
3. $\text{locs}(\tau') = \text{locs}(\tau)$,
4. $\text{duration}(\tau') = \text{duration}(\tau)$, and
5. $|\tau'| \leq F(\mathcal{H}, T) = 24(2\lceil T \rceil + 1) \times |\mathbf{X}|^2 \times |\mathbf{Q}|^2 \times (2\text{cmax} + 3)^{2|\mathbf{X}|}$, where $\text{cmax} := \max\{2, M\}$.

Proof. The result follows from the observation in Section 5.1 about monotonic rectangular automata; \mathcal{H}^ϵ is such an automaton, where the maximum rate of any clock (called rmax in Section 5.1) is bounded by 2. \square

We have now all the results from previous papers that we need to establish the proof of Lemma 58. It is useful to contrast the statements of Lemma 58 and Theorem 56. First, Lemma 58 applies to all trajectories and not just those with a bounded number of steps. Second, in Lemma 58 closeness is measured with respect to J^* (as opposed to trajectories of \mathcal{H}).

Lemma 58. Let \mathcal{H} be a bounded timed automaton with clocks \mathbf{X} . For any distance $\alpha : (0, \frac{1}{2|\mathbf{X}|})$ there is $E : \mathbb{R}_+$ such that for any $\epsilon : [0, E]$ and for any $\tau' : \llbracket \mathcal{H}^\epsilon \rrbracket_*$ such that $\text{first}(\tau') \in J^*$ we have $d_\infty(\text{last}(\tau'), J^*) < \alpha$; here J^* refers to the set returned on Line 9 in Algorithm 3.

Proof. The proof of this lemma is similar to the proof of Theorem 45 in [85] with significant departures. Let M be the maximum constant appearing on constraints of \mathcal{H} . We know M is a bound on clock values in \mathcal{H} . Let W be the number of regions in the region graph of \mathcal{H} (as in line 2). Take $k = F(\mathcal{H}, 6MW + 2M)$, where F is the function from Theorem 36. We will prove that the desired $E = \min\{\frac{1}{3}, E(\alpha, k)\}$, where $E(\alpha, k)$ is from Theorem 56. Fix E to be this value. Notice that for this choice of E , the rate of every variable in \mathcal{H}^ϵ is $\geq 1 - \epsilon > \frac{1}{2}$. This is the reason we choose E to be at most $\frac{1}{3}$.

The proof is by induction on $|\tau'|$. For the base case, consider τ' such that $|\tau'| \leq k$. From Theorem 56 and the definition of E , there is a trajectory $\tau : \llbracket \mathcal{H} \rrbracket_*$ that is α -close to τ' . Since J^* is a closed set, $\text{first}(\tau) \in [\text{first}(\tau')] \subseteq J^*$. Finally, observe that $\text{reach}(\mathcal{G}, J^*) \subseteq J^*$ (\mathcal{G} is the region graph of \mathcal{H}) and so $\text{last}(\tau) \in J^*$. From Theorem 56, we have $d_\infty(\text{last}(\tau), \text{last}(\tau')) < \alpha$ and so $d_\infty(\text{last}(\tau'), J^*) < \alpha$.

For the inductive step, if $|\tau'| > F(\mathcal{H}, \text{duration}(\tau'))$ then Theorem 36 ensures that there is a trajectory $\tau'' \in \llbracket \mathcal{H}^\epsilon \rrbracket_*$ that starts and ends in the same state and has fewer steps. The lemma then follows by the induction hypothesis applied to τ'' .

Let us, therefore, assume that $|\tau'| \leq F(\mathcal{H}, \text{duration}(\tau'))$ and $\text{duration}(\tau') > 6MW + 2M$. Observe that all clocks in \mathcal{H}^ϵ are bounded by M , and the rate of every clock is $\geq 1 - \epsilon > \frac{1}{2}$. So no time transition has duration $2M$ or longer. Thus, without loss of generality, we can write τ' as $\tau' = \tau_{\text{first}} \frown \tau_{\text{last}}$, where $\text{duration}(\tau_{\text{last}})$ is between $6MW$ and $6MW + 2M$. From Theorem 36, there is $\tau'' : \llbracket \mathcal{H}^\epsilon \rrbracket_*$ such that 1. $|\tau''| \leq k = F(\mathcal{H}', 6MW + 2M)$, 2. $\text{duration}(\tau'') = \text{duration}(\tau_{\text{last}})$, 3. $\text{first}(\tau'') = \text{first}(\tau_{\text{last}})$, and 4. $\text{last}(\tau'') = \text{last}(\tau_{\text{last}}) = \text{last}(\tau')$. By Theorem 56 we know there is $\tau : \llbracket \mathcal{H} \rrbracket_*$ that stays α -close to τ'' every step of the way. Let ν_i and ν_i'' be respectively states of τ and τ'' at step i . Applying the induction hypothesis to τ'' , for any $i < |\tau''|$, we have $d_\infty(\nu_i'', J^*) \leq \alpha$. Thus, by triangle inequality, for any $i < |\tau''|$,

$$d_\infty(\nu_i, J^*) \leq d_\infty(\nu_i, \nu_i'') + d_\infty(\nu_i'', J^*) \leq 2\alpha < \frac{1}{|X|}$$

Thus, using Lemma 55, we have $[\nu_i] \cap J^* \neq \emptyset$ for any $i < |\tau''|$.

The previous paragraph establishes the closeness of every state ν_i'' , except possibly the last; we need to show that $\text{last}(\tau_{\text{last}}) = \text{last}(\tau'')$ is close. By construction, we know that $d_\infty(\text{last}(\tau''), \text{last}(\tau)) \leq \alpha$. Thus, proving $\text{last}(\tau) \in J^*$ will complete the induction step.

Next, because \mathcal{H}^ϵ is bounded and the rate of every variable is $> \frac{1}{2}$, every variable is reset within $2M$ time since its last reset. Thus, in any trajectory of duration $\geq 2M$, every variable is reset. Based on these observations, we can partition τ'' as $\tau'' = \tau_1'' \frown \tau_2'' \frown \dots \frown \tau_m''$, where for every i , 1. $\text{duration}(\tau_i'') \leq 2M$, 2. every variable is reset in subtrajectory τ_i'' , and 3. $m \geq 6MW/2M = 3W \geq W + 2$. Since $\text{trace}(\tau) = \text{trace}(\tau'')$, we can partition τ similarly into $\tau = \tau_1 \frown \dots \frown \tau_m$, where every variable is reset in each subtrajectory τ_i . The proof

can now be completed using two technical lemmas that are proved next. We will show

- There is a $j < |\tau|$ such that there is a subtrajectory of τ starting from ν_j that is a progress cycle (Lemma 59).
- $[\nu_j]$ is, in fact, part of a progress cycle of length $\leq W + 2W|\mathbf{X}|$ (Lemma 60).

Assuming the above hold, we can conclude that $\nu_j \in J^*$ because of line 6 of Algorithm 3. Also, since τ is a trajectory of \mathcal{H} , we can conclude that $\nu_i \in J^*$ for all $i \geq j$ (Line 7 of Algorithm 3), completing the proof of the induction step. \square

Lemma 59. Let \mathcal{H} be a timed automaton with W regions in its region graph. Let τ be any trajectory of \mathcal{H} such that

$$\tau = \tau_1 \frown \tau_2 \frown \cdots \frown \tau_m$$

where $m \geq W + 2$ and every variable of \mathcal{H} is reset in each τ_i (for every i). Then there is a subtrajectory of τ that is a progress cycle.

Proof. As $m \geq W + 2$ and the number of regions is bounded by W , by pigeon-hole principle, there must be s, t and region r such that 1. $t \geq s + 2$, and 2. r is visited in both τ_s and τ_t . These observations ensure that there is a subtrajectory of $\tau_s \frown \cdots \frown \tau_t$ that forms a cycle, and since there is an index u between s and t , it is also a progress cycle since all the variables will be reset in τ_u . \square

Lemma 60. Let \mathcal{H} be any timed automaton with \mathcal{G} as its region graph and W as the number of nodes in \mathcal{G} . For any cycle π in \mathcal{G} there is a cycle π' in \mathcal{G} such that

1. $|\pi'| \leq W + 2W|\mathbf{X}|$,
2. $\pi_0 = \pi'_0$ (they start from the same node), and
3. π and π' reset the same set of clocks.

Proof. For any path u and set of paths \mathcal{C} , let $\mathbf{R}(u)$ be the set of variables reset along u and $\mathbf{R}(\mathcal{C}) := \bigcup_{u \in \mathcal{C}} \mathbf{R}(u)$. If $\mathbf{R}(\pi) = \emptyset$ then π' can be any simple cycle that starts from π_0 and resets no variable (because of π we know such cycle

exists). Otherwise, let $\{\pi^1, \dots, \pi^m\}$, for some $m: \mathbb{N}_+$, be the set of simple cycles π is composed of. Start from $\mathcal{C}_0 = \emptyset$ and for any $i > 0$ let

$$\mathcal{C}_i := \begin{cases} \mathcal{C}_{i-1} \cup \{\pi^i\} & \text{if } \mathbf{R}(\pi^i) \not\subseteq \mathbf{R}(\mathcal{C}_{i-1}) \\ \mathcal{C}_{i-1} & \text{otherwise} \end{cases}$$

Clearly $0 < |\mathcal{C}_m| \leq |\mathbf{X}|$ and $\mathbf{R}(\mathcal{C}_m) = \mathbf{R}(\pi)$. We know any two cycles in \mathcal{C}_m can be connected in \mathcal{G} using a path of length at most W . Furthermore, any cycle in \mathcal{C}_m can be connected from/to π_0 using a path of length at most W . Therefore, we can start from π_0 , visit all cycles in \mathcal{C}_m , and come back to π_0 using a cycle of length at most $W + W|\mathbf{X}|$. Adding the total length of simple cycles in \mathcal{C}_m to this number gives us a cycle of length at most $W + 2W|\mathbf{X}|$. \square

We conclude this section by observing that $\text{limreach}^\epsilon(\mathcal{H})$ can be computed in polynomial space.

Theorem 61. For any bounded timed automaton \mathcal{H} , $\text{limreach}^\epsilon(\mathcal{H})$ can be computed by an algorithm that uses space that is bounded by a polynomial function of $|\mathcal{H}|$.

Proof. From Theorem 54, we know that $J^* = \text{limreach}^\epsilon(\mathcal{H})$. If we show that Algorithm 3 uses only polynomial space, then we are done. Unfortunately, as presented, Algorithm 3 uses exponential space and runs in exponential time. But small changes to the presentation of Algorithm 3 establish the complexity bounds stated in the theorem.

First observe that J^* is a union of regions of \mathcal{H} . Therefore, if we show that, given a region r of \mathcal{H} , the problem of determining if $r \in J^*$ can be solved in polynomial space then we will establish the complexity bounds. Based on lines 3 through 7, we conclude that $r \in J^*$ iff there is a path $\pi := \pi_0 \pi_1 \pi_2 \dots \pi_{2m}$, for some $m \leq W$ such that

1. π_0 is an initial region of \mathcal{H}' ,
2. π_{2i+1} is reachable from π_{2i} , for $0 \leq i \leq m$,
3. $[\pi_{2i}] \cap \pi_{2i-1} \neq \emptyset$, for $1 \leq i \leq m$, and
4. for $1 \leq i \leq m$, π_{2i} is on a progress cycle.

By progressively guessing regions on a path/cycle, each of the above conditions can be checked in (nondeterministic) polynomial space, giving us the necessary bounds. \square

5.3.1 Extension to Unbounded Time Automata

Our results for computing $\text{limreach}^\epsilon(\cdot)$ apply to bounded timed automata. However, this is not a restriction as observed by the following proposition.

Proposition 62. Let \mathcal{H} be any timed automaton with locations \mathbf{Q} , variables \mathbf{X} and maximum constant \mathbf{M} . There is a *bounded* timed automaton \mathcal{H}' over the same set of variables \mathbf{X} , locations \mathbf{Q}' with $|\mathbf{Q}'| \leq 2^{|\mathbf{X}|} \times |\mathbf{Q}|$, and maximum constant $\mathbf{M} + 1$, such that for all $\epsilon, \delta : [0, 1)$, $\mathcal{H}_\delta^\epsilon$ and $\mathcal{H}'_\delta^\epsilon$ are weakly bisimilar.

Proof. Let $\mathcal{H} = (\mathbf{Q}, \mathbf{X}, \mathbf{Q}^{\text{init}}, \mathbf{I}, \mathbf{E})$. Observe that clock decreases its value only when it is reset. Thus, if a clock of \mathcal{H} has a value larger than $\mathbf{M} + 1$, its actual value is not important since it will always be larger than $\mathbf{M} + 1$, until it is reset. This intuition is exploited in constructing the bounded timed automaton \mathcal{H}' , which remembers in its location if a clock is larger than $\mathbf{M} + 1$ (and hence its exact value is not important). The formal construction is as follows. $\mathcal{H}' := (\mathbf{Q}', \mathbf{X}, \mathbf{Q}^{\text{init}}_{\mathcal{H}'}, \mathbf{I}', \mathbf{E}')$ where

- $\mathbf{Q}' := \mathbf{Q} \times 2^{\mathbf{X}}$. Intuitively, in any location $(q, S) : \mathbf{Q}_{\mathcal{H}'}$, we have $x \in S$ iff value of x in the corresponding region of \mathcal{H} is $\geq \mathbf{M} + 1$. Therefore, a location (q, S) where $x \in S$ implies that none of the guards of the form $x \leq c$ are satisfied in $\mathcal{H}_\delta^\epsilon$ and all of the guards of the form $x \geq c$ are satisfied in $\mathcal{H}_\delta^\epsilon$.
- $\mathbf{Q}^{\text{init}}_{\mathcal{H}'} := \{(q, \emptyset) \mid q \in \mathbf{Q}^{\text{init}}\}$
- $\mathbf{I}'((q, S), x) := \mathbf{I}(q, x) \cap [0, \mathbf{M} + 1]$
- $\mathbf{E}' := \mathbf{E}_1 \cup \mathbf{E}_2$, where
 - \mathbf{E}_1 is the set of edges of the form $((q_1, S), (q_2, S \setminus r), l, g', r)$ such that $(q_1, q_2, l, g, r) \in \mathbf{E}$ and g' is constructed from g using the following rules: 1. if $x \notin S$ then $g'(x) := g(x)$. 2. if $x \in S$ and $g(x) \subseteq (-\infty, c)$ for some $c : \mathbb{R}$ then $g'(x) := \emptyset$, otherwise 3. $g'(x) := \mathbb{R}$.
 - \mathbf{E}_2 is the set of edges of the form $((q, S), (q, S \cup \{x\}), \emptyset, x \geq \mathbf{M} + 1, \{x\})$.

If the edges of \mathbf{E}_2 are silent/invisible then it is easy to see that \mathcal{H} and \mathcal{H}' are weakly bisimilar. \square

Proposition 62 allows one to compute for any timed automaton \mathcal{H} the set of all regions that contain states of \mathcal{H} that belong to $\text{limreach}^\epsilon(\mathcal{H})$. The algorithm is as follows: Construct \mathcal{H}' , compute $\text{limreach}^\epsilon(\mathcal{H}')$ using Algorithm 3, and then find all the regions that are bisimilar to J^* computed by Algorithm 3. Note that this algorithm will still run in PSPACE because even though \mathcal{H}' has exponentially many locations, the number of regions of \mathcal{H}' is still exponential in $|\mathcal{H}|$.

5.4 Experimental Results

We applied the above algorithmic ideas to verify the robustness of two protocols. Models of both protocols are available in UPPAAL. The first one we considered is Fischer’s mutual exclusion protocol for n processes. Any mutual exclusion protocol must satisfy at least three properties: 1. no two processes enter a critical section at the same time, 2. there are no deadlocks in the system, and 3. any request to access a critical section will eventually be granted. The first property is a safety property and the next two are liveness properties. Each of these properties can be verified to hold for the unperturbed protocol using UPPAAL for $n \leq 6$ in less than 2 seconds. Using UPPAAL, we could also show that the system satisfies each of these properties when there are 6 processes and guards are enlarged by $\delta := 0.01$ ⁶; the verification time for this was also less than 2 seconds. Therefore, using Lemma 42 and knowing the maximum constant in this model is 3, the model is also robust with $\epsilon := \frac{0.01}{12}$ and $\delta := \frac{0.01}{2}$.

The next example we verified, is 2Doors. It involves two doors and two users that interact using the following rules: 1. a room has two doors which cannot be opened at the same time, 2. a door starts to open if its button is pushed, 3. it takes six seconds for a door to open, and thereafter it stays open for at least four seconds, but no more than eight seconds, 4. it takes six seconds for a door to close and it stays closed for at least five seconds. We checked the following properties: 1. Mutex: The two doors are never open at the same time. 2. Either of doors can be opened. 3. Liveness: Whenever a button is pushed, the corresponding door will eventually open. 4. The

⁶ The model has a strict inequality $x > k$. We first replace that with $x \geq k + 1$ and positively verified that all properties are still satisfied.

system is deadlock-free. Using UPPAAL, in less than 2 seconds we could show that the system satisfies each of these properties when guards are enlarged by $\delta:=0.0001$. Therefore, using Lemma 42 and knowing the maximum constant in this model is 8, the model is also robust with $\epsilon:=\frac{0.0001}{32}$ and $\delta:=\frac{0.0001}{2}$.

Note that properties we checked in these two examples are not necessarily expressible using the special type of (robust) ω -regular model checking problems we have considered in this chapter. However, Lemma 42 establishes language inclusion between executions of \mathcal{H} under different types of perturbation, regardless of the property one might want to check.

5.5 Related Work

Robust model checking was first considered in [36, 37]. The authors first defined a distance function on executions of a hybrid automaton \mathcal{H} . Then for *robust* safety analysis they added one restriction: an execution τ is robustly inside the reachable set of states in \mathcal{H} iff for some $\epsilon : \mathbb{R}_+$, all executions in the ϵ -neighborhood of τ are in the reachable set of states in \mathcal{H} . They proved that under this constraint, the safety problem for timed automata stays PSPACE-complete, and safety problem for rectangular automata stays undecidable.

In [38], the author has another definition for robustness. He considers noise as an inevitable part of every physical environment. Therefore, if A is an *exact* reachable set after a continuous transition, he considers an ϵ -ball around it to be reachable. The author proves that under this notion of robustness, if the reachable set is bounded then iterative computation of the reach set (as we did in Chapter 4) always reaches a fixed-point. Otherwise, if the reachable set is unbounded but the unsafe set is bounded then we can always terminate the search if we found no unsafe state to be reachable after a predetermined finite number of steps, and announce the input system safe. Finally, to obtain a decision procedure, he specifies continuous dynamics using first order formulas on reals with addition and multiplications. He does not use ODEs to specify continuous dynamics of the system.

In [39] authors consider a similar perturbation for the class of piecewise constant derivative hybrid automata. In this class of hybrid automata, there are finitely many polyhedra as invariants and dynamics of every variable is defined using a constant ODE that only depends on the polyhedra the current

state is inside it. Like [38], here authors also prove that although the exact safety problems are undecidable for this class of hybrid automata, the robust safety problem is in fact decidable.

In [40] authors use ideas in [73, 94, 95] and show that if the actual answer to the safety problem does not depend on a small *numerical* perturbation then not only the *bounded time* reachability problem for a large class of non-linear hybrid automata becomes decidable, under very mild assumptions, the complexity of this problem will be PSPACE-complete.

All the research on robust model checking mentioned so far, only considers the complexity of robust safety model checking and completely ignore the implementability of designs for which the safety and in general any required property φ is proved. To overcome the implementability issue for timed automata, [41] first proposes a *program semantics*. In this semantics, it is assumed that the hardware executes the following procedure in an infinite loop: 1. it first reads value of the only global clock and evaluate all the guards in the current location, and 2. if there was any enabled transition then one of them will be taken non-deterministically. It is assumed that this procedure takes at most Δ_L units of time. It is also assumed that the clock is updated at least every Δ_P units of time. Furthermore, the clock may drift by $\epsilon: \mathbb{R}_{\geq 0}$ which means after delay $d: [0, \Delta_P]$, value of the global clock will be increased by a value in $[d(1 - \epsilon), d(1 + \epsilon)]$. The input timed automaton \mathcal{H} is called *implementable* with respect to property φ iff, for some $\Delta_L, \Delta_P, \epsilon: \mathbb{R}_+$, all executions of the implementation that satisfies the new parametric semantics satisfy φ . For the purpose of verification, [41] introduces another semantics for timed automata called *Almost-ASAP semantics* which is obtained by first dividing the system into two hybrid automata: a controller and a plant. Then guards of the controller are enlarged by Δ . It was then proved that if $\epsilon = 0$ and $\Delta > 3\Delta_L + 4\Delta_P$ then for any ω -regular property φ , $\mathcal{H}_\Delta^\epsilon \models \varphi$ implies $\text{Prg}_{\Delta_L, \Delta_P, \epsilon}(\mathcal{H}) \models \varphi$, which simply implies implementability. Note that in $\mathcal{H}_\delta^\epsilon$, we do not divide the system into two hybrid automata. Therefore, $\mathcal{H}_\delta^\epsilon$ simulates the semantics defined in [41].

Finally, the implementability issue is being looked at differently in [96]. In most models, non-integer variables are all of type reals (\mathbb{R}). But in the implementation, they could be encoded as finite-precision floating point variables. In [96], authors consider an implementation of a model and bound the distance between reachable states that is caused by finite-precision floating

point variables in the implementation. The model is considered robust iff this difference is small enough.

5.6 Conclusions

In this chapter, we investigated the robust model checking problem for monotonic rectangular automata. We also considered timed automata without the progress cycle assumption, when only clocks can drift and when both guards and clocks are perturbed. We proved time bounded robust reachability analysis of monotonic rectangular automata, when only flows are perturbed, is NEXPTIME-complete. We used this to prove our next two results. All the other results focus on timed automata where guards are expressed using non-strict inequalities. These automata neither satisfy the progress cycle assumption, nor are assumed to have bounded variables. We then proved that the exact reachable set of timed automata with infinitesimally perturbed clocks can be computed (optimally) in polynomial space. We also proved the following two problems are PSPACE-complete: Model checking ω -regular properties of timed automata with only drifted clocks (note that this does not subsume our previous result), and Model checking ω -regular properties of timed automata with both drifted clocks and enlarged guards. Furthermore, we proved robust satisfaction of an ω -regular property of timed automata with only enlarged guards implies robust satisfaction of the same property when only clocks drift, and is equivalent to robust satisfaction of the property when clocks and guards are perturbed. Finally, we show that safety under clock drifts does not imply safety under enlarged guards. An application of our results is robust model checking of partially synchronized protocols. Our results do not yield an algorithm to compute $\limreach_{\delta}^{\epsilon}(\mathcal{H})$ ($= \limreach_{\delta}(\mathcal{H})$), which is a possible direction for future exploration. Furthermore, robust model checking of subclasses of metric temporal logics is another direction of future exploration.

Chapter 6

Conclusions

In Chapter 3 we considered initialized linear inclusion automata and proved that while unbound time reachability problem is undecidable for this class of hybrid automata, the time bounded reachability problem is decidable and is PSPACE-hard. In Chapter 4 we considered unbounded time safety model checking of non-linear polyhedral automata, which is a much larger class of hybrid automata, at the expense of losing decidability. We introduced a sound but incomplete abstraction refinement algorithm for this problem and proved its progress. We also implemented this algorithm in a tool called **HARE** and compared it with seven other tools on many examples. Our results show that **HARE**'s performance is better than state-of-the-art tools for safety model checking affine hybrid automata and comparable with state-of-the-art tools for safety model checking non-linear polyhedral automata. In Chapter 5 we considered robust model checking for monotonic rectangular automata when flows are perturbed and for timed automata when guards and/or flows are perturbed. We established the following results in this context. Time bounded robust reachability analysis of monotonic rectangular automata, when only flows are perturbed, is NEXPTIME-complete. We proved the exact reachable set of timed automata under infinitesimal perturbation to clocks and without the progress cycle assumption can be computed (optimally) in polynomial space. We also proved the following problems are PSPACE-complete — Model checking ω -regular properties of timed automata when only clocks drift (note that this does not subsume our previous result), and model checking ω -regular properties of timed automata when both guards and clocks are perturbed. Furthermore, we proved that the robust satisfaction of an ω -regular property of timed automata with only enlarged guards implies the robust satisfaction of the same property with only clock drifts, and is equivalent to the robust satisfaction of the property when clocks and guards are perturbed. Finally, we showed that the safety under clock

drifts does not imply the safety under enlarged guards.

6.1 Future Directions

There are several directions for possible improvements and future work. In this section, we present each of these directions.

6.1.1 Improvements

For finding the exact reachable set in Chapter 3, the most important open question is to find a bound on verifying $a \ln \frac{b}{c} < d$, for integers a, b, c, d . This would be a significant result with consequences beyond what we have in Chapter 3. For example, while satisfiability check for polynomial exponential problems is known to be decidable [97], to the best of our knowledge, there is still no complexity known for this problem due to the unknown complexity of deciding $a \ln \frac{b}{c} < d$. The exact same problem exists for model checking continuous time Markov chains [98] and reachability analysis in linear dynamical systems [99].

For model checking using abstraction refinement in Chapter 4, there are two improvements one can consider. The first one is to use some optimization techniques to find a better abstraction for non-linear dynamics. Currently, **HARE** uses interval analysis for abstracting non-linear dynamics and this could give very conservative results. We believe nonlinear convex optimization can be helpful in this case. The second improvement is to replace **dReach** with some other tools like **C2E2** or **FLOW***. In our experiment, **dReach** performance decreases dramatically when we move to non-linear dynamics. We hope using tools written specifically for computing reachable sets for non-linear dynamics can help improve performance.

For robust model checking in Chapter 5, there are at least two directions of future exploration one can consider. The first one is to solve the robust model checking problem (even safety model checking) for timed automata when only clocks drift and constraints can be strict. To the best of our knowledge there is only one paper that deals with the restricted version of this problem [100] and almost no result in this paper has a precise proof. The second one is to give a more efficient algorithm to compute reachable set

of timed automata when clocks drift by infinitesimally small amount. Note that as far as the theory is concerned, Algorithm 3 in Chapter 5 is optimal. But this is not necessarily an efficient algorithm as already observed for its predecessor in [85].

6.1.2 New Avenues

Abstraction Refinement Engine. A very interesting and surprisingly challenging research idea could be extending HARE to support full linear-time temporal logic (LTL) model checking. The problem is extremely undecidable and one needs to be very careful about the right abstraction refinement. That being said, the necessity of checking progress conditions, *e.g.* something good eventually happen to the system is undeniable (consider medical devices for example).

Robust Model Checking. A promising approach to deal with the current difficulties in model checking cyber-physical systems is *robust* model checking. This is where we address issues like uncertainty in design parameters, validity of synthesis after small changes in parameters, small noise in physical environment, impreciseness in sensors, distributed controllers, implementability of verified designs, and complexity of model checking. We have already considered the implementability issue for a simple class of models in [101] and extended the current results in multiple directions. But there are a lot more one can do. For example, one can consider 1. robust model checking of timed automata where only dynamics and guards related to a subset of variables are perturbed, 2. robust model checking of a composition of multiple timed automata, 3. robust model checking of metric temporal logic (MTL), using robust model checking of timed automata, and 4. robust model checking of more expressive classes of hybrid automata. But the more basic and still more vital and interesting question is what is the right definition for robust model checking. For example, to the best of our knowledge, there is no accepted definition for robust model checking of MTL. Also, what will happen if adapt the robustness definition that is used in δ -complete decision procedures [94,95] (*i.e.* write the semantics of a formula as a first order logic formula, and then perturbed it as in δ -complete decision procedures). Does this help to lower complexity? Does the resulting definition still gives

interesting/useful problems? We would like to explore robustness in more detail. Even though, Robust Control is an active field of research in Control Theory, surprisingly, not much has been done in robust verification of cyber-physical systems. We would like to understand the underlying properties of cyber-physical systems in connection with different notions of robustness. Why some robustness definitions make the corresponding model checking problem a lot easier, some make them harder, and yet some do not change the complexity.

Statistical Model Checking and its Relation to Robust Model Checking. Probabilistic hybrid automata are widely used in modeling cyber-physical systems. What makes them more interesting is the mesmerizing and still very little known relation between robust and statistical model checking of hybrid automata. Statistical model checking has proven itself to be very useful and scalable in model checking cyber-physical systems. In [102] we transformed a deterministic system into a stochastic one in order to make model checking even possible. In [103] we did even more. We transformed a stochastic problem into a deterministic one first, and then used stochastic techniques to solve the deterministic problem. In these and some of our other papers like [104, 105], our results heavily rely on some notion of robustness. We would like to explore the relation between robustness and the ability to go back and forth between deterministic and stochastic problems/algorithms. Were our experiences in [102–105] just two lucky accidents or, for example, whenever a problem is robust, under some right notion of robustness, can we always transform a deterministic *robust* problem into a *stochastic* one and solve it more efficiently?

References

- [1] T. A. Henzinger, “The theory of hybrid automata,” in *Proceeding of IEEE Symposium on Logic in Computer Science*, 1996, pp. 278–292.
- [2] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, “What’s decidable about hybrid automata?” in *Journal of Computer and System Sciences*. ACM Press, 1995, pp. 373–382.
- [3] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, “The algorithmic analysis of hybrid systems,” *TCS*, vol. 138, no. 1, pp. 3–34, 1995.
- [4] E. Asarin, O. Maler, and A. Pnueli, “Reachability analysis of dynamical systems having piecewise-constant derivatives,” *TCS*, vol. 138, no. 1, pp. 35–65, 1995.
- [5] V. Mysore and A. Pnueli, “Refining the undecidability frontier of hybrid automata,” in *FSTTCS*, 2005, pp. 261–272.
- [6] V. Vladimerou, P. Prabhakar, M. Viswanathan, and G. Dullerud, “STORMED hybrid systems,” in *ICALP*, 2008, pp. 136–147.
- [7] O. Maler and A. Pnueli, “Reachability analysis of planar multi-linear systems,” in *CAV*, 1992, pp. 194–209.
- [8] E. Asarin, G. Schneider, and S. Yovine, “Algorithmic analysis of polygonal hybrid systems, Part I: Reachability,” *TCS*, vol. 379, no. 1-2, pp. 231–265, 2007.
- [9] E. Asarin, G. Schneider, and S. Yovine, “On the decidability of the reachability problem for planar differential inclusions,” in *HSCC*, 2001, pp. 89–104.
- [10] P. Prabhakar, V. Vladimerou, M. Viswanathan, and G. Dullerud, “A decidable class of planar linear hybrid systems,” in *HSCC*, 2008, pp. 401–414.
- [11] R. Alur and D. L. Dill, “A theory of timed automata,” *TCS*, vol. 126, pp. 183–235, 1994.

- [12] G. Lafferriere, G. Pappas, and S. Sastry, “o-minimal hybrid systems,” *MCSS*, vol. 13, pp. 1–21, 2000.
- [13] J. Ouaknine, A. Rabinovich, and J. Worrell, “Time-bounded verification,” in *CONCUR’09*, ser. LNCS, M. Bravetti and G. Zavattaro, Eds., vol. 5710. Springer-Verlag, Sep. 2009, pp. 496–510.
- [14] M. Jenkins, J. Ouaknine, A. Rabinovich, and J. Worrell, “Alternating timed automata over bounded time,” *Logic in Computer Science, Symposium on*, vol. 00, pp. 60–69, 2010.
- [15] T. Brihaye, L. Doyen, G. Geeraerts, J. Ouaknine, J. F. Raskin, and J. Worrell, “On reachability for hybrid automata over bounded time,” in *ICALP (2)*, 2011, pp. 416–427.
- [16] T. Brihaye, L. Doyen, G. Geeraerts, J. Ouaknine, J. F. Raskin, and J. Worrell, “Time-bounded reachability for monotonic hybrid automata: Complexity and fixed points.” in *ATVA*, ser. LNCS, vol. 8172. Springer, 2013, pp. 55–70.
- [17] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, “Counterexample-guided abstraction refinement,” in *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, 2000, pp. 154–169.
- [18] T. Ball and S. Rajamani, “Bebop: A symbolic model checker for Boolean programs,” in *Proc. of the SPIN*, 2000, pp. 113–130.
- [19] G. Holzmann and M. Smith, “Automating software feature verification,” *Bell Labs Technical Journal*, vol. 5, no. 2, pp. 72–87, 2000.
- [20] J. Corbett, M. Dwyer, J. Hatcliff, S. Laubach, C. Pasareanu, Robby, and H. Zheng, “Bandera: Extracting finite-state models from Java source code,” in *ICSE*, 2000, pp. 439–448.
- [21] T. Henzinger, R. Jhala, R. Majumdar, and G. Sutre, “Lazy Abstraction,” in *POPL 2002*, 2002, pp. 58–70.
- [22] R. Alur, T. Dang, and F. Ivančić, “Predicate abstraction for reachability analysis of hybrid systems,” *ACM Trans. Embed. Comput. Syst.*, vol. 5, no. 1, pp. 152–199, 2006.
- [23] E. Clarke, A. Fehnker, Z. Han, B. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald, “Verification of Hybrid Systems Based on Counterexample-Guided Abstraction Refinement,” in *TACAS*, 2003, pp. 192–207.

- [24] E. Clarke, A. Fehnker, Z. Han, B. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald, “Abstraction and Counterexample-Guided Refinement in Model Checking of Hybrid Systems,” *JFCS*, vol. 14, no. 4, pp. 583–604, 2003.
- [25] M. Sorea, “Lazy approximation for dense real-time systems,” in *Proc. of FORMATS/FTRFTS*, 2004, pp. 363–378.
- [26] A. Fehnker, E. Clarke, S. Jha, and B. Krogh, “Refining Abstractions of Hybrid Systems using Counterexample Fragments,” in *HSCC 2005*, 2005, pp. 242–257.
- [27] H. Dierks, S. Kupferschmid, and K. Larsen, “Automatic Abstraction Refinement for Timed Automata,” in *FORMATS*, 2007, pp. 114–129.
- [28] M. Segelken, “Abstraction and Counterexample-guided Construction of Omega-Automata for Model Checking of Step-discrete linear Hybrid Models,” in *CAV*, 2007, pp. 433–448.
- [29] S. K. Jha, B. H. Krogh, J. E. Weimer, and E. M. Clarke, “Reachability for linear hybrid automata using iterative relaxation abstraction,” in *HSCC 2007*, 2007, pp. 287–300.
- [30] V. Gupta, T. Henzinger, and R. Jagadeesan, “Robust timed automata,” in *Proceedings of the International Workshop on Hybrid and real-time Systems*, 1997, pp. 331–345.
- [31] F. Cassez, T. Henzinger, and J.-F. Raskin, “A comparison of control problems for timed and hybrid systems,” in *Proceedings of HSCC*, 2002, pp. 134–148.
- [32] M. D. Wulf, L. Doyen, N. Markey, and J.-F. Raskin, “Robustness and implementability of timed automata,” in *Proceedings of FORMATS*, 2004, pp. 118–133.
- [33] M. D. Wulf, L. Doyen, and J.-F. Raskin, “Almost ASAP semantics: From timed models to timed implementations,” *Formal Aspects of Computing*, vol. 17, no. 3, pp. 319–341, 2005.
- [34] K. Altisen and S. Tripakis, “Implementation of timed automata: An issue of semantics or modeling?” in *Proceedings of FORMATS*, 2005, pp. 273–288.
- [35] R. Alur, S. L. Torre, and P. Madhusudan, “Perturbed timed automata,” in *Proceedings of HSCC*, 2005, pp. 70–85.
- [36] V. Gupta, T. A. Henzinger, and R. Jagadeesan, *Robust timed automata*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 331–345.

- [37] T. A. Henzinger and J.-F. Raskin, *Robust Undecidability of Timed and Hybrid Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 145–159.
- [38] M. Fränzle, *Analysis of Hybrid Systems: An Ounce of Realism Can Save an Infinity of States*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 126–139.
- [39] E. Asarin and A. Bouajjani, “Perturbed turing machines and hybrid systems,” in *LICS*, 2001, pp. 269–278.
- [40] S. Kong, S. Gao, W. Chen, and E. Clarke, *dReach: δ -Reachability Analysis for Hybrid Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 200–205.
- [41] M. De Wulf, L. Doyen, and J.-F. Raskin, *Almost ASAP Semantics: From Timed Models to Timed Implementations*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 296–310.
- [42] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, “SpaceEx: Scalable verification of hybrid systems,” in *Proc. 23rd International Conference on Computer Aided Verification*, 2011.
- [43] G. Frehse, “Phaver: Algorithmic verification of hybrid systems past hytech,” in *Hybrid Systems: Computation and Control, 8th International Workshop, HSCC 2005, Zurich, Switzerland, March 9-11, 2005, Proceedings*, 2005, pp. 258–273.
- [44] S. Bogomolov, G. Frehse, M. Greitschus, R. Grosu, C. S. Pasareanu, A. Podelski, and T. Strump, “Assume-guarantee abstraction refinement meets hybrid systems,” in *10th International Haifa Verification Conference*, 2014, pp. 116–131.
- [45] S. Ratschan and Z. She, “Safety verification of hybrid systems by constraint propagation based abstraction refinement,” *ACM Transactions in Embedded Computing Systems*, vol. 6, no. 1, 2007.
- [46] P. S. Duggirala, S. Mitra, M. Viswanathan, and M. Potok, *C2E2: A Verification Tool for Stateflow Models*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 68–82.
- [47] X. Chen, E. Ábrahám, and S. Sankaranarayanan, *Flow*: An Analyzer for Non-linear Hybrid Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 258–263.
- [48] P. Duggirala and A. Tiwari, “Safety verification for linear systems,” in *Proceedings of EMSOFT*, 2013.

- [49] T. A. Henzinger, P. H. Ho, and H. Wong-Toi, “Algorithmic analysis of nonlinear hybrid systems,” *Automatic Control, IEEE Transactions on*, vol. 43, no. 4, pp. 540–554, apr 1998.
- [50] J. Reif and S. Tate, “On threshold circuits and polynomial computation,” *SIAM Journal on Computing*, vol. 21, no. 5, pp. 896–908, 1992.
- [51] W. Hesse, E. Allender, and D. Barrington, “Uniform constant-depth threshold circuits for division and iteratd multiplication,” *Journal of Computer and System Sciences*, vol. 65, no. 4, pp. 695–716, 2002.
- [52] R. P. Brent, “Fast algorithms for high-precision computation of elementary functions (invited talk),” in *Seventh Conference on Real Numbers and Computers (RNC7)*, no. 7-8, 10-12 July 2006.
- [53] J. S. Miller, “Decidability and complexity results for timed automata and semi-linear hybrid automata,” in *Proceedings of the Third International Workshop on Hybrid Systems: Computation and Control*, ser. HSCC ’00. Springer-Verlag, 2000, pp. 296–309.
- [54] R. Alur, R. Kurshan, and M. Viswanathan, “Membership questions for timed and hybrid automata,” in *In RTSS’98*. Press, 1998, pp. 254–263.
- [55] R. Chadha, D. Kini, and M. Viswanathan, “Quantitative information flow in boolean programs,” in *Proceedings of POST*, 2014.
- [56] C. Yap, “Pi is in log space,” <http://www.cs.nyu.edu/exact/doc/pi-log.pdf>.
- [57] A. Casagrande, C. Piazza, A. Policriti, and B. Mishra, “Inclusion dynamics hybrid automata,” *Inf. Comput.*, vol. 206, no. 12, pp. 1394–1424, Dec. 2008.
- [58] M. Fränzle, “Analysis of hybrid systems: An ounce of realism can save an infinity of states,” in *CSL, volume 1683 of LNCS*. Springer, 1999, pp. 126–140.
- [59] O. Botchkarev and S. Tripakis, “Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations,” in *Proceedings of the Third International Workshop on Hybrid Systems: Computation and Control*, ser. HSCC ’00. London, UK, UK: Springer-Verlag, 2000, pp. 73–88.
- [60] M. Agrawal and P. S. Thiagarajan, “The discrete time behavior of lazy linear hybrid automata,” in *HSCC*, ser. Lecture Notes in Computer Science, M. Morari and L. Thiele, Eds., vol. 3414. Springer, 2005, pp. 55–69.

- [61] R. Gentilini, “Reachability problems on extended O-minimal hybrid automata,” in *Formal Modeling and Analysis of Timed Systems (FORMATS)*, ser. LNCS, P. Pettersson and W. Yi, Eds., vol. 3829. Uppsala, Sweden: Springer, 2005, pp. 162–176.
- [62] N. Roohi and M. Viswanathan, *FORMATS*. Cham: Springer International Publishing, 2014, ch. Time-Bounded Reachability for Initialized Hybrid Automata with Linear Differential Inclusions and Rectangular Constraints, pp. 191–205.
- [63] N. Roohi and M. Viswanathan, “Time-bounded reachability for initialized hybrid automata with linear differential inclusions and rectangular constraints,” University of Illinois at Urbana-Champaign, Tech. Rep., 2014, <http://hdl.handle.net/2142/49952>.
- [64] P. Prabhakar, P. S. Duggirala, S. Mitra, and M. Viswanathan, “Hybrid automata-based CEGAR for rectangular hybrid systems,” in *VMCAI*, 2013, pp. 48–67.
- [65] N. Roohi, P. Prabhakar, and M. Viswanathan, “Hybridization based CEGAR for hybrid automata with affine dynamics,” in *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, 2016, pp. 752–769.
- [66] A. Puri, P. Varaiya, and V. Borkar, “ ϵ -approximation of differential inclusions,” 1995.
- [67] T. A. Henzinger, P. H. Ho, and H. Wong-Toi, “Hytech: a model checker for hybrid systems,” *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 1, pp. 110–122, 1997.
- [68] X. Chen, E. Ábrahám, and S. Sankaranarayanan, “Flow*: An analyzer for non-linear hybrid systems,” in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, N. Sharygina and H. Veith, Eds. Springer Berlin Heidelberg, 2013, vol. 8044, pp. 258–263.
- [69] R. Bagnara, P. M. Hill, and E. Zaffanella, “The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems,” *Science of Computer Programming*, vol. 72, no. 1–2, pp. 3–21, 2008.

- [70] L. De Moura and N. Bjørner, “Z3: An efficient smt solver,” in *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, ser. TACAS’08/ETAPS’08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 337–340.
- [71] “Boost Interval Arithmetic Library,” http://www.boost.org/doc/libs/1_62_0/libs/numeric/interval/doc/interval.htm, Accessed: 2016-10-19.
- [72] A. Puri, V. S. Borkar, and P. Varaiya, “Epsilon-approximation of differential inclusions,” in *Hybrid Systems III: Verification and Control*, 1995, pp. 362–376.
- [73] S. Gao, S. Kong, and E. M. Clarke, *dReal: An SMT Solver for Non-linear Theories over the Reals*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 208–214.
- [74] M. Kalicinski and S. Redl, “Boost Property Tree,” 2016. [Online]. Available: http://www.boost.org/doc/libs/1_62_0/doc/html/property_tree.html
- [75] N. Roohi, P. Prabhakar, and M. Viswanathan, “HARE: A Hybrid Abstraction Refinement Engine for Verifying Non-Linear Hybrid Automata,” to be appeared in Tools and Algorithms for the Construction and Analysis of Systems (TACAS) 2017.
- [76] S. Bogomolov, A. Donze, G. Frehse, R. Grosu, T. T. Johnson, H. Ladan, A. Podelski, and M. Wehrle, “Guided search for hybrid systems based on coarse-grained space abstractions,” *International Journal on Software Tools for Technology Transfer*, Oct. 2014.
- [77] T. T. Johnson, J. Green, S. Mitra, R. Dudley, and R. S. Erwin, “Satellite rendezvous and conjunction avoidance: Case studies in verification of nonlinear hybrid systems,” in *Proceedings of the 18th International Conference on Formal Methods (FM 2012)*, 2012.
- [78] A. Fehnker and F. Ivancic, “Benchmarks for hybrid systems verification,” in *In Hybrid Systems: Computation and Control*, 2004, pp. 326–341.
- [79] L. Doyen, T. Henzinger, and J.-F. Raskin, “Automatic rectangular refinement of affine hybrid systems,” in *Formal Modeling and Analysis of Timed Systems*, ser. Lecture Notes in Computer Science, P. Pettersson and W. Yi, Eds. Springer Berlin Heidelberg, 2005, vol. 3829, pp. 144–161.

- [80] J. Nellen, E. Ábrahám, and B. Wolters, “A CEGAR tool for the reachability analysis of PLC-controlled plants using hybrid automata,” in *Formalisms for Reuse and Systems Integration*, 2015, vol. 346, pp. 55–78.
- [81] A. Zutshi, J. V. Deshmukh, S. Sankaranarayanan, and J. Kapinski, “Multiple shooting, CEGAR-based falsification for hybrid systems,” in *Proceedings of the 14th International Conference on Embedded Software*, 2014.
- [82] K. Larsen, P. Pettersson, and W. Yi, “UPPAAL in a nutshell,” *International Journal on Software Tools for Technology Transfer*, vol. 1, pp. 134–152, 1997.
- [83] S. Yovine, “KRONOS: A verification tool for real-time systems,” *International Journal on Software Tools for Technology Transfer*, vol. 1, pp. 123–133, 1997.
- [84] A. Puri, “Dynamical properties of timed automata,” *Discrete Event Dynamic Systems*, vol. 10, no. 1-2, pp. 87–113, 2000.
- [85] M. Wulf, L. Doyen, N. Markey, and J.-F. Raskin, “Robust safety of timed automata,” *Formal Methods in System Design*, vol. 33, no. 1, pp. 45–84, 2008.
- [86] P. Bouyer, N. Markey, and O. Sankur, “Robust model-checking of timed automata via pumping in channel machines,” in *Formal Modeling and Analysis of Timed Systems, FORMATS*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 97–112.
- [87] G. Behrmann, A. Fehnker, T. Hune, K. Larsen, P. Pettersson, J. Romijn, and F. Vaandrager, “Minimum-cost reachability for priced timed automata,” in *Proceedings of HSCC*, 2001, pp. 147–161.
- [88] P. Bouyer and F. Chevalier, “On conciseness of extensions of timed automata,” *J. Autom. Lang. Comb.*, vol. 10, no. 4, pp. 393–405, Apr. 2005.
- [89] P. Bouyer, N. Markey, and P.-A. Reynier, “Robust model-checking of linear-time properties in timed automata,” in *Proceedings of LATIN*, 2006, pp. 238–249.
- [90] N. Karmarkar, “A new polynomial-time algorithm for linear programming,” *Combinatorica*, vol. 4, no. 4, pp. 373–395, 1984.
- [91] A. R. Bradley and Z. Manna, *The Calculus of Computation: Decision Procedures with Applications to Verification*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

- [92] J. Robinson, “The undecidability of algebraic rings and fields,” in *American Math Society* 10, 1959, pp. 950–957.
- [93] M. Ben-Or, D. Kozen, and J. Reif, “The complexity of elementary algebra and geometry,” in *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, ser. STOC ’84. ACM, 1984, pp. 457–464.
- [94] S. Gao, J. Avigad, and E. M. Clarke, “Delta-decidability over the reals,” in *LICS*. IEEE Computer Society, 2012, pp. 305–314.
- [95] S. Gao, J. Avigad, and E. M. Clarke, “ δ -complete decision procedures for satisfiability over the reals,” in *IJCAR*. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 286–300.
- [96] E. Goubault, S. Putot, P. Baufreton, and J. Gassino, “Static analysis of the accuracy in control systems: Principles and experiments,” in *Proceedings of the 12th International Conference on Formal Methods for Industrial Critical Systems*, ser. FMICS’07. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 3–20.
- [97] M. Achatz, S. McCallum, and V. Weispfenning, “Deciding polynomial-exponential problems,” in *Proceedings of the Twenty-first International Symposium on Symbolic and Algebraic Computation*, ser. ISSAC ’08. New York, NY, USA: ACM, 2008, pp. 215–222.
- [98] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton, “Model-checking continuous-time markov chains,” *ACM Trans. Comput. Logic*, vol. 1, no. 1, pp. 162–170, July 2000.
- [99] E. Hainry, “Reachability in linear dynamical systems,” in *Proceedings of the 4th Conference on Computability in Europe: Logic and Theory of Algorithms*, ser. CiE ’08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 241–250.
- [100] C. Dima, *Dynamical Properties of Timed Automata Revisited*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 130–146.
- [101] N. Roohi, P. Prabhakar, and M. Viswanathan, “Robust Model Checking of Timed Automata under Clock Drifts,” To be appeared in Hybrid Systems: Computation and Control (HSCC) 2017.
- [102] N. Roohi, Y. Wang, M. West, G. Dullerud, and M. Viswanathan, “Statistical Verification of the Toyota Powertrain Control Verification Benchmark,” To be appeared in Hybrid Systems: Computation and Control (HSCC) 2017.

- [103] Y. Wang, N. Roohi, M. West, M. Viswanathan, and G. Dullerud, “A Mori-Zwanzig and MITL Based Approach to Statistical Verification of Continuous-time Dynamical Systems,” *International Federation of Automatic Control (IFAC PapersOnLine)*, vol. 48, no. 27, pp. 267–273, 2015.
- [104] Y. Wang, N. Roohi, M. West, M. Viswanathan, and G. Dullerud, “Verifying Continuous-Time Stochastic Hybrid Systems Via Mori-Zwanzig Model Reduction,” in *IEEE Conference on Decision and Control (CDC)*, To be appeared in 2017.
- [105] Y. Wang, N. Roohi, M. West, M. Viswanathan, and G. E. Dullerud, “Statistical Verification of Dynamical Systems using Set Oriented Methods,” in *Hybrid Systems: Computation and Control (HSCC)*, 2015, pp. 169–178.