Scalable Synthesis and Verification: Towards Reliable Autonomy

Thesis by

Sumanth Dathathri

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California, USA

2020

Defended 2 March 2020

To my mom, Shruthi and the memory of my father.

Acknowledgments

I have had the joy of working with and being mentored by Richard M. Murray, my adviser. This thesis would not have been possible without Richard's continued support. I thank him for his patience and encouragement, even at times when progress was slow. I am also extremely grateful for his support with independently pursuing my research interests and collaborations, including spending time away from Caltech. I would like to extend my deepest thanks to Joel Burdick for his mentorship through my early graduate school years, and for providing me with office space alongside his group. I would also like to express my gratitude to Chandy for being a part of my committee and sharing insightful feedback. I am grateful for the patient collaboration Sicun Gao offered even when I floundered around with research ideas, and for being a part of my committee. He has also been a wonderful mentor to me.

I thank Dj and Yisong Yue for being extremely supportive and understanding collaborators and mentors. I sincerely appreciate their unwavering support, even as I flailed around with research directions. Jason Yosinski, always full of cheerful encouragement, has been the most amazing mentor and collaborator. I also wish to thank Nikos Aréchiga for his collaboration and mentorship. I am grateful for extremely rewarding collaborations with Yuxiao Chen and Ioannis Filippidis, both of whom taught me a great deal. I thank Scott C. Livingston for being an amazing collaborator during my early graduate school years, for teaching me a great deal about research, and for his continued friendship.

I am thankful to my friends Ellen, Tung, Richard C., Andrea, Vijay Chav, Ravi, Kishore (KJ), Pushkar, Neel, Sisir, Kamyar, Shubika, Sophia, Gautam, Jialin and many others (too many to list here) for making graduate school thoroughly enjoyable, and for their unflinching support during the highs and the lows. I am indebted beyond measure to my violin teacher,

Cynthia Fogg, whose weekly lessons were a source of tremendous comfort during the most difficult times. I am also indebted to the Caltech squash club and Caltech cricket team for some very cherishable memories.

Abstract

We have seen the growing deployment of autonomous systems in our daily life, ranging from safety-critical self-driving cars to dialogue agents. While impactful and impressive, these systems do not often come with guarantees and are not rigorously evaluated for failure cases. This is in part due to the limited scalability of tools available for designing correct-by-construction systems, or verifying them posthoc. Another key limitation is the lack of availability of models for the complex environments with which autonomous systems often have to interact with. In the direction of overcoming these above mentioned bottlenecks to designing reliable autonomous systems, this thesis makes contributions along three fronts.

First, we develop an approach for parallelized synthesis from linear-time temporal logic Specifications corresponding to the generalized reactivity (1) fragment. We begin by identifying a special case corresponding to singleton liveness goals that allows for a decomposition of the synthesis problem, which facilitates parallelized synthesis. Based on the intuition from this special case, we propose a more generalized approach for parallelized synthesis that relies on identifying equicontrollable states.

Second, we consider learning-based approaches to enable verification at scale for complex systems, and for autonomous systems that interact with black-box environments. For the former, we propose a new abstraction refinement procedure based on machine learning to improve the performance of nonlinear constraint solving algorithms on large-scale problems. For the latter, we present a data-driven approach based on chance-constrained optimization that allows for a system to be evaluated for specification conformance without an accurate model of the environment. We demonstrate this approach on several tasks, including a lane-change scenario with real-world driving data.

Lastly, we consider the problem of interpreting and verifying learning-based compo-

nents such as neural networks. We introduce a new method based on Craig's interpolants for computing compact symbolic abstractions of pre-images for neural networks. Our approach relies on iteratively computing approximations that provably overapproximate and underapproximate the pre-images at all layers. Further, building on existing work for training neural networks for verifiability in the classification setting, we propose extensions that allow us to generalize the approach to more general architectures and temporal specifications.

Published Content and Contributions

Y. Chen, S. Dathathri, T. Phan-Minh, and R. M. Murray. Counter-example Guided Learning of Bounds on Environment Behavior. page arXiv:2001.07233, Jan 2020.
S.D participated in the formulation of the project, development of the method, and the efforts for the writing of the manuscript.

S. Dathathri, J. Welbl, K. D. Dvijotham, R. Kumar, A. Kanade, J. Uesato, S. Gowal, P.-S. Huang, and P. Kohli. Scalable neural learning for verifiable consistency with temporal specifications, 2020. URL: `https://openreview.net/forum?id=BklC2RNKDS`.
S.D led the formulation of the project, development of the method, all experiments and the efforts for the writing of the manuscript.

S. Dathathri, S. Gao, and R. M. Murray. Inverse abstraction of neural networks using symbolic interpolation. In *The Thirty-Third AAAI Conference on Artificial Intelligence*, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence, Honolulu, Hawaii, USA, January 27 - February 1, 2019, pages 3437–3444. AAAI Press, 2019. DOI: 10.1609/aaai.v33i01.33013437.
S.D led the formulation of the project, development of the method, all experiments and the efforts for the writing of the manuscript.

S. Dathathri, I. Filippidis, and R. M. Murray. Parallelizing synthesis from temporal logic specifications by identifying equicontrollable states. In *Robotics Research*, pages 827–842, Cham, 2020. Springer International Publishing. ISBN 978-3-030-28619-4.
S.D led the formulation of the project, development of the method, all experiments and the efforts for the writing of the manuscript.

S. Dathathri and R. M. Murray. Decomposing GR(1) games with singleton liveness guar-

antees for efficient synthesis. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 911–917, DOI: 10.1109/CDC.2017.8263775.
S.D led the formulation of the project, development of the method, all experiments and the efforts for the writing of the manuscript.

S. Dathathri, N. Arechiga, S. Gao, and R. M. Murray. Learning-based abstractions for nonlinear constraint solving. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, IJCAI-17, pages 592–599, 2017. DOI: 10.24963/ijcai.2017/83.
S.D led the formulation of the project, development of the method, all experiments and the efforts for the writing of the manuscript.

N. Aréchiga, S. Dathathri, S. Vernekar, N. Kathare, S. Gao, and S. Shiraishi. Osiris: A tool for abstraction and verification of control software with lookup tables. In *Proceedings of the 1st International Workshop on Safe Control of Connected and Autonomous Vehicles*, SCAV'17, pages 11–18, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349765. DOI: 10.1145/3055378.3055384.
S.D participated in the formulation of the project, development of the method, all experiments and the efforts for the writing of the manuscript.

Contents

Chapter 1

Introduction

Recent years have seen an increased presence of robots, autonomous vehicles, and other cyber-physical systems in our daily lives [9]. The increasing presence of complex cyber-physical systems, particularly in safety-critical applications, demands that we are able to mitigate and guarantee the absence of undesired behavior. The complexity of these systems lends itself to non-trivial failure modes, which are often difficult to detect and debug, and can have catastrophic consequences when they occur in safety-critical systems (e.g., autonomous driving [1]). The rise in such safety-critical applications and their increasing complexity makes a strong case for the wider adoption of formal methods tools and algorithms to help design reliable systems with formal guarantees. Despite the rising need for such tools, several bottlenecks exist that currently limit the adoptability of formal methods tools in the design/verification of autonomous systems at scale.

The prohibitive computational cost associated with existing tools for the design (e.g., synthesis from temporal logic specifications) and verification (e.g., SAT/SMT solvers) of autonomous systems is a key barrier limiting their wider adoption [86]. Another key bottleneck is the challenge in developing reliable and precise models for the behavior of the cyber-physical. Recent years have seen an increased presence of robots, autonomous vehicles, and other cyber-physical systems in our daily lives[9]. The increasing presence of complex cyber-physical systems, particularly in safety-critical applications, demands that we are able to mitigate and guarantee the absence of undesired behavior. The complexity of these systems lends itself to non-trivial failure modes, that are often difficult to detect and debug, and can have catastrophic consequences when they occur in safety critical

systems (e.g., autonomous driving [1]). The rise in such safety-critical applications and their increasing complexity makes a strong case for the wider adoption of formal methods tools and algorithms to help design reliable systems with formal guarantees. Despite the rising need for such tools, several bottlenecks exist that currently limit the adoptability of formal methods tools in the design/verification of autonomous systems at scale.

The prohitive computational cost associated with existing tools for the design (e.g. syntstems, and the often complex and uncertain environments they interact with. Accurate models are often a precursor to being able to leverage existing tools for the design of these systems.

A new frontier is the emergence of autonomous systems with complex learning-enabled components. However, the progress accompanying this emergence towards guaranteeing the formal correctness of learning-enabled systems at large has been quite limited. As learning-enabled systems continue to accomplish impressive feats [36, 96, 104], it is essential that we continue to develop tools that will enable the analysis of these systems, and ensure their reliability.

This thesis makes contribution towards the above discussed bottlenecks. We first present results aimed at enabling the parallelized synthesis of controllers from specifications in linear temporal logic (LTL) [94]. Temporal logic is an expressive language that was initially developed to express complex properties desired from software programs. More recently, temporal logic has found increasing use in expressing tasks for cyber-physical systems for specifications that go beyond simple classical point-to-point motion planning [24, 38, 80, 88]. For specifications in LTL, we present results that allow for the parallelized synthesis of controllers from specifications. The majority of the focus here is directed towards planning for discrete-transition systems.

Constraint solvers (e.g., z3 [37], dReal [53]), which can reason over arbitrary formulas in first-order logic, have emerged as key tools towards being able to provide formal guarantees regarding the reliability of autonomous cyber-physical systems. However, often, the constraint satisfaction problems that arise corresponding to the verification/synthesis of cyber-physical systems are challenging and computationally intractable. Besides constraint solvers, temporal logic falsification tool-boxes such as S-Taliro [5], Breach[39] are another

key set of tools that have proven useful in debugging/designing reliable systems. These tools can be leveraged when we have access to systems that are simulatable or can be modeled precisely. However, we often require our controllers to function in environments that are difficult to model or simulate – for example, an autonomous car that interacts with human-driven cars and pedestrians. In such instances, it is difficult to directly leverage these tools. To overcome these challenges, we propose approaches for learning abstractions that i) facilitate the scaling of constraint solvers, and ii) enable us to use tools like S-Taliro, while providing probabilistic guarantees regarding the correctness of the abstractions.

Going beyond software programs and control systems, which are the primary focus of existing formal methods tools, we develop new tools and algorithms for the analysis and design of reliable machine learning systems. We propose i) an algorithm that allow us to automatically abstract complex neural network-based learned components into simpler symbolic formulas, and ii) a method for training neural networks such that their temporal logic constraints can be guaranteed in a tractable manner. Below, we introduce in more detail the key contributions of this thesis.

1.1 Main Contributions

The results presented in the thesis can be loosely characterized into three parts:

- algorithms for parallelized synthesis from LTL specifications
- approaches for learning abstractions to assist verification with constraint solvers and falsification tools and
- algorithms for the design and analysis of learned components, with a focus on neural networks.

1.1.1 Scalable Synthesis Through Parallelization

Chapter 2 builds on existing approaches for control synthesis from temporal logic specifications, to develop new extensions that facilitate parallelized synthesis. Parallelized

synthesis, unlike current algorithms that are primarily sequential, exploits the structure of the synthesis problem to decompose the primary synthesis into independent subproblems that can be solved separately. The solution from these subproblems can then be utilized to synthesize a global controller that solves the primary synthesis problem. The work in Chapter 2 is based on [32] and [35], which is a result of collaboration with Ioannis Filippidis and Richard M. Murray. In [32], a special sub-class of LTL specifications is identified where synthesis can be trivially parallelized, allowing for scalable synthesis. Both the correctness and the soundness of synthesis using the decomposition-based approach for this sub-class of specifications is proved in [32]. The work in [35] extends this approach to more general specifications using the notion of *equicontrollable classes* (sets of states with bidirectional reachability). The approach from [35], while being sound, loses the completeness guarantees. The resulting improvements from both approaches introduced in [32, 35] in comparison to existing approaches is demonstrated on robot motion planning benchmarks.

1.1.2 Learning-Based Abstractions for Verification

In Chapter 3, we introduce two key ideas based on [6, 23, 33] that demonstrate how learning-based abstractions can help existing tools for debugging controllers be more widely applicable. First, we begin by developing an approach that enables the abstraction of lookup table-based components, which are common place in industrial cyber-physical systems. The key idea is to replace the complex lookup tables with simpler abstraction that come with guarantees (such that they are provable overapproximations). These abstractions can then be used to speed up verification for the cyber-physical system using constraint solvers. We demonstrate the approach a cruise control benchmark [124]. Beyond cyber-physical systems with lookup table, these constraint solvers play a key-role in providing formal guarantees for diverse applications, ranging from planning and scheduling for control systems to program analysis. This is based on [6], a result of collaboration with Nikos Aréchiga, Shashank Vernekar, Nagesh Kathare, Sicun Gao, and Shinichi Shiraishi.

In [33], which is a result of collaboration with Nikos Aréchiga, Sicun Gao and Richard

M. Murray, we generalize the approach for verifying lookup table and develop a more general learning-based approach that can abstract and simplify hard constraint problems by replacing the original CSP with a simpler (provable) relaxation. The latter part of the chapter (based on [23], which is a result of collaboration with Yuxiao Chen, Tung Phan-Minh, and Richard M. Murray) introduces a approach for learning data-driven abstractions that allow us to formally guarantee the correctness of controllers interacting with difficult to model, uncertain environments with high probability. The learned abstractions can assist tools such as S-Taliro in debugging systems before they are deployed in the real-world.

1.1.3 Interpreting and Verifying Neural Networks

In Chapter 4, we consider the analysis of systems with learned neural networks. Neural networks can represent complex nonlinear functions, and it is often difficult to interpret their behavior and formally guarantee correctness. In the first part of the chapter we introduce an approach that decomposes learned neural networks into modules, and computes abstractions for these modules that are likely to have a simpler structure. These abstractions can then be used for interpreting and verifying the behavior of the neural network. This part of the chapter is result of a collaboration with Sicun Gao and Richard M. Murray [34].

In the second part of the chapter, we introduce an approach for training neural netweorks wherein the verification objective expressed in temporal logic is folded into the training objective. We demonstrate the applicability of the approach on tasks from across domains (language processing, image captioning, and reinforcement learning).

Finally, Chapter 5 concludes the thesis by discussing open problems and promising avenues for extending the results presented in this thesis. This part of the chapter is based on work at DeepMind in collaboration with Krishnamurthy (Dj) Dvijotham, Ramana Kumar, Aditya Kanade, Jonathan Uesato, Johannes Welbl, Po-Sen Huang, and Pushmeet Kohli.

Chapter 2

Parallelized Synthesis for LTL Specifications

Logic specifications assist in the design of complex systems by enabling us to precisely specify desired behavior for the system. In particular, reactive synthesis from LTL specifications for robotics applications has received increased attention [24, 38, 80, 88], where the controller is synthesized considering all possible behaviors of the environment. *Reactivity* here implies that the synthesized controller *reacts* to the environments behavior while deciding the control action. The inherent structure of this problem, where the system has to reason over all admissible environment behavior makes synthesis algorithms prohibitively computationally expensive. The scalability of these algorithms has been identified as a bottleneck in their adoption [86]. In this chapter, we introduce algorithms for parallelized synthesis from LTL algorithms to help with scalability.

Synthesizing finite-memory strategies from LTL specifications for the general case is doubly exponential in the length of the formula [95], but for *generalized reactivity (1)* (GR (1)) – a rich, expressive fragment of LTL – the synthesis can be done in polynomial time in the number of states and the number of liveness guarantees for the system and the number of liveness assumptions for the adversary [76]. GR(1) specifications model a game where the system and its adversary infinitely often satisfy a set of liveness constraints while making moves that satisfy certain safety constraints. This fragment in particular has received considerable attention since its conception because of the computational tractability associated with it. The GR(1) fragment is also particularly attractive because of the symbolic

nature of the synthesis algorithm, which enables scaling to large finite-transition systems. First, we begin by identifying a special sub-class of GR(1) synthesis problems with singleton liveness guarantees, where the problem can be trivially decomposed and synthesis can be parallelized. Building on this intuition, we introduce the notion of equicontrollability and propose an approach that generalizes the approach beyond specifications with singleton liveness guarantees.

2.1 Preliminaries

In this section, we briefly introduce the notation that we use. Additional details and precise definitions can be found in [10, 94].

Atomic propositions are statements that evaluate to `True` or `False`. Consider a finite set of atomic propositions AP. Denote by Σ the set of states of the system ($\Sigma := \mathcal{P}(\text{AP})$, where $\mathcal{P}(\text{AP})$ is the power-set of AP). We denote the restriction of the set $\mathcal{X}$ to $\mathcal{Y}$ by $\mathcal{X}_{|\mathcal{Y}}$, i.e., $\mathcal{X}_{|\mathcal{Y}} = \mathcal{X} \cap \mathcal{Y}$.

We write $s \models p$ if a state $s \in \Sigma$ satisfies a proposition $p \in \text{AP}$. A state $s \in \Sigma$ satisfies a proposition $p \in \text{AP}$ if and only if $p \in s$. We will work with the Boolean operators $\wedge$ (conjunction), $\vee$ (disjunction), $\rightarrow$ (implication) and $\leftrightarrow$ (bi-implication) to construct Boolean formulas. The temporal operators we use are *next* ($\bigcirc$), *eventually* ($\Diamond$) and *always* ($\Box$). For a Boolean formula ξ over AP, by $[\![\xi]\!]$ we refer to the set of states satisfying ξ. The semantics of LTL are defined over infinite strings in Σ^ω. For a string $\sigma \in \Sigma^\omega$, and some $t \in \mathbb{N}$, the satisfaction of an LTL formula beginning at time t is denoted by $\sigma, t \models \varphi$. When $\sigma, 0 \models \varphi$ (i.e., $t = 0$), we simply write $\sigma \models \varphi$. For ease of notation, we further extend the semantics of LTL to reason over finite strings. For a finite string $\gamma \in \Sigma$, we define: $\gamma \models \rho \leftrightarrow \gamma\alpha| = \rho$ for any $\alpha \in \Sigma^\omega$. For $\sigma \in \Sigma^\omega$, σ_k refers to the $(k+1)^{\text{th}}$ element in the sequence σ with σ_0 being the first element.

For reactive synthesis, we model the synthesis problem as a two-player game where the environment satisfies certain assumptions on its behavior and with these assumptions being satisfied, the system is required to behave in a desired manner while reacting to the environment. To formulate the reactive synthesis problem, we first partition AP into two

disjoint sets of variables $\mathrm{AP_e}$ and $\mathrm{AP_a}$ such that the set $\mathrm{AP_e}$ is controlled by the environment and $\mathrm{AP_a}$ is controlled by the agent being designed. The sets $\mathrm{AP_e}$, $\mathrm{AP_a}$ form a partition of AP, i.e., $\mathrm{AP} = \mathrm{AP_e} \cup \mathrm{AP_a}$ and $\mathrm{AP_e} \cap \mathrm{AP_a} = \emptyset$. Define the state spaces over these sets of propositions as $\Sigma_\mathrm{e} := \mathcal{P}(\mathrm{AP_e})$ and $\Sigma_\mathrm{a} := \mathcal{P}(\mathrm{AP_a})$.

The synthesis problem is to find a function $f : (\Sigma_\mathrm{e} \times \Sigma \times M) \to (\Sigma_\mathrm{a} \times M)$ such that the sequences of states generated by this strategy satisfy a given specification φ. M is a finite set of memory values with a unique initial memory value $\underline{m}$, in other words f is a finite-memory strategy. For a finite-memory strategy f, the set of infinite sequences that occur when using f are referred to as *plays*:

$$\begin{aligned}\mathrm{Plays}(f) = \{\sigma \in \Sigma^\omega | \exists m \in M^\omega \text{ such that } m_0 = \underline{m} \text{ and} \\ \forall k \geq 0.(\sigma^\mathrm{a}_{k+1}, m_{k+1}) = f(\sigma^\mathrm{e}_{k+1}, \sigma_k, m_k)\}.\end{aligned} \tag{2.1}$$

A strategy is *winning* for a formula φ if and only if all plays of f satisfy the formula and it is *input enabled*, meaning that f should be defined at the initial state-memory pair, as well as at any state-memory pair that can be reached in any play. A state $s \in \Sigma$ is a *winning state* for a specification if there exists a strategy that is winning with the given state s as the initial state. The maximal set of all such winning states is the *winning set* for a specification. For a specification φ, we denote its winning set as W_φ. A specification is *realizable* if there exists there is a winning strategy from the given initial state.

2.1.1 Generalized Reactivity (1)

For reactive synthesis, we focus specifically on the GR(1) fragment. The GR(1) specification models a two-player game where the controlled agent has to satisfy a set of liveness guarantees and safety constraints under some assumed behavior for the environment. This assumed behavior for the environment in turn consists of a set of liveness properties and safety constraints. A GR(1) formula has the form:

$$\varphi := (\theta^\mathrm{e} \wedge \theta^\mathrm{a}) \wedge (\square\rho^\mathrm{e} \wedge \bigwedge_{j=1}^{m} \square\diamond\psi^\mathrm{e}_j) \to (\square\rho^\mathrm{a} \wedge \bigwedge_{i=1}^{m} \square\diamond\psi^\mathrm{a}_i), \tag{2.2}$$

where θ^{e} is a Boolean function of propositions in AP and marks the set of assumed initial poses for the environment, The set of valid initial poses for the for the controlled agent is described by the Boolean formula θ^{a}. The assumed safety behavior for the environment (ρ^{e}) is a Boolean function of propositions in $\mathrm{AP}_a \cup \mathrm{AP}_e \cup \bigcirc \mathrm{AP}_e$, with

$$\bigcirc \mathrm{AP}_e = \{\bigcirc \alpha : \alpha \in \mathrm{AP}_e\}.$$

The specification for valid actions for the controlled agent (ρ^{a}) is a Boolean function of $\mathrm{AP}_a \cup \mathrm{AP}_e \cup \bigcirc \mathrm{AP}_e \cup \bigcirc \mathrm{AP}_a$, while ψ^{a}, ψ^{e} are Boolean functions of $\mathrm{AP}_a \cup \mathrm{AP}_e$. A GR(1) synthesis problem is to find a strategy f that is winning for this formula and, in addition, the following must hold for the every play σ of the strategy:

$$\sigma \models \boxminus \rho^{e} \rightarrow \boxminus \rho^{a}, \tag{2.3}$$

where $\boxminus$ is the *historically* temporal operator [89]. This ensures that the agent does not violate its safety constraint by forcing the environment to violate its assumption in the future.

Definition 1. *Given transition rules for the controlled agent (ρ^{a}) and the environment transition (ρ^{e}), the set of* reachable states *(Σ^{reach}) is the set of states in Σ that can be visited through any sequence of valid actions for the environment and the controlled agent.*

Formally,

$$\Sigma_G^{\mathrm{reach}} = \{v | \exists X \in \Sigma_e^*, \exists Y \in \Sigma_a^* \text{ such that } (X_0, Y_0) \models \theta, v = (X_{-1}, Y_{-1}), |X| = |Y| \text{ and } \forall k < |X| - 1.(X_k, Y_k, X_{k+1}) \models \rho^{e}, (X_k, Y_k, X_{k+1}, Y_{k+1}) \models \rho^{a}\},$$

where Σ^* is the Kleene closure of Σ and $|X|$ is the length of the sequence X. The set of reachable states can be computed in at most $O(|\Sigma|)$ symbolic steps.

Synthesis for GR(1) Specifications Synthesis for GR(1) specifications can be performed by solving a μ-calculus formula with an alternation depth of three [76] . For the case with two liveness guarantees, the μ-calculus formula in [76] can be written using the *vector*

notation as

$$\nu \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} \begin{bmatrix} \mu Y \left(\bigvee_{j=1}^{m} \nu X \left(\left(\left(\psi_1^{\mathrm{a}} \wedge \bigcirc Z_2 \right) \vee \bigcirc Y \right) \vee \neg \psi_j^{\mathrm{e}} \wedge \bigcirc X \right) \right) \\ \mu Y \left(\bigvee_{j=1}^{m} \nu X \left(\left(\left(\psi_2^{\mathrm{a}} \wedge \bigcirc Z_1 \right) \vee \bigcirc Y \right) \vee \neg \psi_j^{\mathrm{e}} \wedge \bigcirc X \right) \right) \end{bmatrix} \tag{2.4}$$

Here, ν is the greatest fixpoint operator and μ is the least fixpoint operator (see [101] for detailed definitions of these operators). Intuitively, the fixed point in X characterizes the set of states from which the system can force the play to stay indefinitely in $[\![\neg \psi_j^{\mathrm{e}}]\!]$ for some j or in a finite number of steps reach a state satisfying $\psi^{\mathrm{s}} \vee \bigcirc Y$. Staying in $[\![\neg \psi_j^{\mathrm{e}}]\!]$ for some j indefinitely implies blocking the environment from satisfying one of its liveness assumptions. The intermediate least fixed point in Y makes sure that the phase of play represented by $\bigcirc Y$ eventually ends in $[\![\psi^{\mathrm{s}}]\!]$. This way either $\diamond \psi^{\mathrm{s}}$ is satisfied or $\bigvee_{i=1}^{m} \diamond \Box \neg \psi_i^{\mathrm{e}}$ is satisfied. The outer greatest-fixed point in νZ_i corresponds to computing the set of states for which we can guarantee that after satisfying φ_i^{s}, the play can be forced to a state satisfying $\varphi_{i \oplus 1}^{\mathrm{s}}$ (or indefinitely stays in states satisfying $\neg \varphi_j^{\mathrm{e}}$ for some j).

The GR(1) fragment is often used for high-level reasoning because of the polynomial-time symbolic algorithms available for the synthesis of strategies for this fragment. Symbolic algorithms allow for reasoning about problems with very large state spaces because they construct strategies by manipulating sets of states, as opposed to an enumerative approach where all the states are stored and searched. For the algorithm outlined in [15] for GR(1) synthesis, the sets are stored and manipulated as binary decision diagrams (BDDs). BDDs serve as compact representations of sets, but the variable ordering can have a significant effect on their size [10].

The complexity for reordering of BDDs is often not taken into account while analyzing the complexity of symbolic synthesis algorithms [15]. Finding the optimal variable ordering that minimizes the size of reduced order BDDs is NP-hard [16]. For a brief introduction to BDDs and their use in symbolic model checking we refer the reader to [10].

The synthesis algorithm outlined in [15] and its implementation in modern solvers [47] results in cubic time algorithms for solving the nested fixpoints. However, using ideas from [18], the nested fixpoints can be solved in quadratic time but this also results in the storing and reordering of $nm|\Sigma|^2$ BDDs (in the worst case).

2.1.2 Reachability Games

Let φ^{e} be the assumption on the behavior of the environment and ρ^{a} be the set of transition rules for the controlled agent. Note that for a GR(1) game, φ^{e} has the form:

$$\varphi^{\mathrm{e}} := \theta^{\mathrm{e}} \wedge \Box\rho^{\mathrm{e}} \wedge \bigwedge_{j=1}^{m} \Box\Diamond\psi_j^{\mathrm{e}}.$$

A *reachability game* can be defined based on the following LTL specification:

$$\varphi^{\mathrm{e}} \rightarrow \Box\rho^{\mathrm{a}} \wedge \left(\bigvee_{s\in\mathcal{B}} \Diamond s\right). \tag{2.5}$$

Definition 2. *For a set of states $\mathcal{B} \subseteq \Sigma$, we denote the set of winning states for the condition in equation* (2.5) *as* WinSet*($\mathcal{B}$), or alternatively as* WinSet *of $\mathcal{B}$.*

In other words, *WinSet*($\mathcal{B}$) for a set $\mathcal{B}$ is the set of states from where the agent can force the system to transition into $\mathcal{B}$ for all admissible behavior for the environment. We shall refer to the synthesis problem corresponding to equation (2.5) as a *reachability game*. Note that this *WinSet* is different from the *winning set* introduced earlier.

Synthesis for reachability games Synthesis for reachability games can be performed by solving a μ-calculus formula with an alternation depth of two [76]. Consider the μ-calculus formula:

$$\mu_{\mathrm{rg}} := \mu Y\left(\bigvee_{j=1}^{m} \nu X\left(\left((\psi^{\mathrm{a}} \vee \textcircled{\Diamond} Y) \vee \neg\psi_j^{\mathrm{e}}\right) \wedge \textcircled{\Diamond} X\right)\right). \tag{2.6}$$

The μ-calculus formula in equation (2.5) corresponds to the two inner fixed points in the μ-calculus formula for the GR(1) game (equation (2.4)). Computing the *WinSet* for $\mathcal{B} \subseteq \Sigma$ takes at most $O(m|\Sigma|^2)$ symbolic steps [76].

2.2 GR(1) specifications With Singleton Liveness Goals

In this section, we first identify a special sub-class of GR(1) specifications that are trivially parallelizable, based on [32]. Consider GR(1) specifications where the liveness goals

correspond to singleton sets, i.e, $|[\![\psi_i^{\mathrm{a}}]\!]| = 1$ for $i = 1, 2, \ldots, n$. The solution to a GR(1) game with n liveness goals can be obtained by combining the solutions to $n+1$ independent reachability games, each involving solving a μ-calculus formula with an alternation depth of 2.

The reachability games are independent, unlike the cyclic dependency in equation (2.4) between the various liveness guarantees. The outermost fixed point computation in Z_i can be avoided here as the liveness guarantees correspond to singleton sets and this allows for the separation of the sub-games (we prove this later). For example, for a GR(1) game with two liveness goals (as in equation (2.4)), the GR(1) game can be split into three reachability games:

$$\psi_1^{\mathrm{a}} \wedge \left(\Box\rho^e \wedge \bigwedge_{i=1}^{m} \Box\Diamond\psi_i^{\mathrm{e}}\right) \rightarrow \left(\Box\rho^a \wedge \Box\Diamond\psi_1^a\right), \tag{2.7}$$

$$\left((\theta^{\mathrm{e}} \wedge \theta^{\mathrm{a}}) \vee \psi_1^{\mathrm{a}}\right) \wedge \left(\Box\rho^{\mathrm{e}} \wedge \bigwedge_{i=1}^{m} \Box\Diamond\psi_i^{\mathrm{e}}\right) \rightarrow \Diamond\psi_2^{\mathrm{a}},$$

$$(\theta^{\mathrm{e}} \wedge \theta^{\mathrm{a}}) \wedge \left(\Box\rho^{\mathrm{e}} \wedge \bigwedge_{i=1}^{m} \Box\Diamond\psi_i^{\mathrm{e}}\right) \rightarrow \Diamond\,\texttt{False}\,.$$

Recall that $\theta^a \wedge \theta^e$ is the initial condition for the original synthesis problem. In general, for a problem with n liveness constraints, the reachability games can be set up as for $j \in \{1, 2, \ldots, n\}$:

$$\theta_j \wedge \left(\Box\rho^e \wedge \bigwedge_{i=1}^{m} \Box\Diamond\psi_i^{\mathrm{e}}\right) \rightarrow \left(\Box\rho^a \wedge \Box\Diamond\psi_{j\oplus 1}^{\mathrm{a}}\right) \tag{2.8}$$

with the initial conditions being $\theta_j = \psi_j^{\mathrm{a}}$ for $j \neq n$ and $\theta_j = \psi_j^{\mathrm{a}} \vee (\theta^e \wedge \theta^a)$ for $j = n$. Note that $\oplus$ is the `modulo` n operator, i.e., $j \oplus 1 = (j+1)$ modulo n. For example, $n \oplus 3 = 3$ when $3 < n$. We shall refer to the winning condition

$$\theta_{k\oplus -1} \wedge \left(\Box\rho^e \wedge \bigwedge_{i=1}^{m} \Box\Diamond\psi_i^{\mathrm{e}}\right) \rightarrow \left(\Box\rho^a \wedge \Box\Diamond\psi_k^{\mathrm{a}}\right)$$

as φ_k^{reach}. Additionally, define φ_0^{reach} as

$$(\theta^e \wedge \theta^a) \wedge \left(\square \rho^e \wedge \bigwedge_{i=1}^{m} \square \diamond \psi_i^{\text{e}} \right) \rightarrow \diamond \texttt{False}. \tag{2.9}$$

Note that for any given state, a strategy that is winning against this condition can only do so by forcing the play to block the environment from satisfying its assumptions.

Combining Strategies from Reachability Games The reachability games φ_k^{reach} for $k = 0, 1, 2, \ldots, n$ are independent and can be solved in parallel. We now formalize how to utilize the strategies obtained from solving these reachability games. We can observe that if φ_0^{reach} is winnable, then from solving φ_0^{reach} we have a strategy for $\bar{\varphi}$ and this is also winning for φ, since the environment is blocked from satisfying its assumptions.

Suppose φ_0^{reach} is not winnable, and the other n reachability games are winnable. We construct the strategy f_G^{φ} by combining the n reachability games such that f_G^{φ} is winning against φ. To do this, we introduce a variable $\mathcal{Z}_n$ that can take values in $\{1, 2, \ldots, n\}$ to track which liveness guarantees have been satisfied in the current cycle, with $\mathcal{Z}_n$ initialized to n. Let $f^{\text{reach}_j} : M^j \times \Sigma \times \Sigma_e \rightarrow M^j \times \Sigma_a$ be the winning strategy for $\varphi_{j \oplus 1}^{\text{reach}}$, with m_0^j as the initial memory. The strategy f_G^{φ} is constructed such that starting with a state $s \models \theta$, the execution follows f^{reach_n} to reach a state satisfying ψ_1^{a} or blocks the environment from satisfying one of the liveness assumptions. If the execution reaches ψ_1^{a}, the strategy switches to f^{reach_1} and reaches ψ_2^{a} or blocks the environment, and so on.

Formally, the strategy:

$$f_G^{\varphi} : (M \times \{1, 2, \ldots, n\}) \times \Sigma \times \mathcal{P}(\text{AP}_{\text{e}}) \rightarrow (M \times \{1, 2, \ldots, n\}) \times \mathcal{P}(\text{AP}_{\text{a}})$$

is constructed as

$$f_G^{\varphi}((w, \mathcal{Z}_n), s, s' \cap \text{AP}_{\text{e}}) = ((w', \mathcal{Z}_n'), s' \cap \text{AP}_{\text{a}}),$$

where if $s \models \psi_{\mathcal{Z}_n \oplus 1}^{\text{a}}$,

$$\mathcal{Z}_n' = \mathcal{Z}_n \oplus 1,$$

$$(w', s' \cap \mathrm{AP_a}) = f^{\mathrm{reach}_{\mathcal{Z}'_n}}(m_0^{\mathcal{Z}'_n}, s, s' \cap \mathrm{AP_e},),$$

and if $s \not\models \psi^{\mathrm{a}}_{\mathcal{Z}_n \oplus 1}$,

$$(w', s' \cap \mathrm{AP_a}) = f^{\mathrm{reach}_{\mathcal{Z}_n}}(w, s, s' \cap \mathrm{AP_e}),$$

$$\mathcal{Z}'_n = \mathcal{Z}_n.$$

Here $\mathcal{Z}'_n$ denotes the value of $\mathcal{Z}_n$ at the next step. Similarly, s' is the next state with s being the current state. When $s \models \psi^{\mathrm{a}}_{\mathcal{Z}_n \oplus 1}$ for a given $\mathcal{Z}_n$, we increment $\mathcal{Z}_n$. Thereby switching to the strategy $f^{\mathrm{reach}_{\mathcal{Z}_n \oplus 1}}$, which we follow till we reach $\psi^{\mathrm{a}}_{\mathcal{Z}_n \oplus 2}$.

If for the initial condition θ, $\varphi_0^{\mathrm{reach}}$ is not winnable and for some i such that $n \geq i > 0$, $\varphi_i^{\mathrm{reach}}$ is not winnable then φ is not winnable from θ.

Results Here, we argue the soundness and the completeness of the decomposition and the constructed strategy. Define $\bar{\varphi}$ as the following formula:

$$\bar{\varphi} := (\theta^e \wedge \theta^a) \wedge \left(\Box\rho^e \wedge \bigwedge_{i=1}^{m} \Box\Diamond\psi_i^{\mathrm{e}} \right) \rightarrow \left(\Box\rho^a \wedge \Diamond\psi_1^{\mathrm{a}} \wedge \left(\bigwedge_{i=1}^{n} \Diamond \left(\psi_i^{\mathrm{a}} \rightarrow \Diamond\psi_{i\oplus 1}^{\mathrm{a}} \right) \right) \right). \quad (2.10)$$

Claim 3. $W_\varphi = W_{\bar{\varphi}}$ if $|[\![\psi_i^{\mathrm{a}}]\!]| = 1 \; \forall i \in \{1, 2 \ldots n\}$.

The winning sets for the formulas $\bar{\varphi}$ and φ(as defined in 2.2) are the same. This implies that the set of states from which there exists a strategy to satisfy each ψ_i^{a} once is the same as the set of states from which there exists a strategy to cycle through each ψ_i^{a} infinitely often. A proof of the claim is provided in Appendix A.1.

Lemma 4. *A GR(1) winning condition of the form*

$$\varphi = (\theta^a \wedge \theta^e) \wedge \left(\Box\rho^e \wedge \bigwedge_{i=1}^{m} \Box\Diamond\psi_i^{\mathrm{e}} \right) \rightarrow \left(\Box\rho^a \wedge \bigwedge_{i=1}^{n} \Box\Diamond\psi_i^{\mathrm{a}} \right)$$

can be solved by solving $n + 1$ *independent reachability games if* $|[\![\psi_i^{\mathrm{a}}]\!]| = 1 \; \forall i \in \{1, 2, \ldots, n\}$.

A proof of the Lemma is provided in Appendix A.2. Note that the construction of the

combined strategy above combines constructions from the proofs of Claim 3 and Lemma 4.

2.2.1 Counterexample for the Non-Singleton Case

While sound, the approach is not complete when than one state can satisfy any of the liveness guarantees. This is because the GR(1) game might not be realizable from all states corresponding to a liveness guarantee, and it might be possible to avoid states corresponding to a liveness guarantee that are not realizable and yet satisfy the specification. In this case, one of the reachability games arising from the decomposition introduced in this section would not be realizable, and would lead to a false negative when the GR(1) game itself is realizable.

Consider the simple environment depicted in Figure 2.1. Let `Office`, `Living Room`, `Music Room`, ... be propositional variables corresponding to the position of the robot in the various rooms. These propositions are assigned values based on the position of the robot. For instance, the variable `Office` is assigned the value `True` when the robot is in the *office* part of the workspace. `Door` and `Door1` are propositional variables that take the value `True` when the corresponding door is open, and `False` otherwise. The problem is to synthesize a strategy for a robot such that it satisfies the liveness goals $\square\diamond$(`Living Room` $\vee$ `Office`) and $\square\diamond$`Garage` with the liveness assumption for the environment being $\square\diamond$`Door`. The robot can move between rooms if the slit connecting them is not blocked. There is no assumption on the behavior of the `Door1`. Here, starting from the *office* (which satisfies the formula `Living Room`$\vee$`Office`), the robot cannot reach the *garage* (the other liveness goal) without assuming that *door1* opens, and this would cause the proposed decomposition-based approach to indicate that the specification is not realizable. However, this is not correct as the robot can choose to avoid visiting the *office* entirely, and cycle between the *living room* and the *garage*, satisfying the liveness goals. We will introduce a more general approach in the subsequent section that can handle such scenarios.

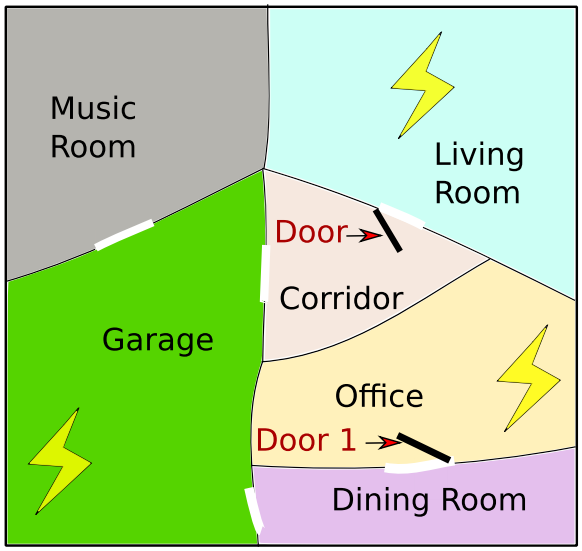


Figure 2.1: Example Workspace

2.3 Parallelized Synthesis by Identifying Equicontrollable States

Here, we build on the intuition from the previous section to generalize the approach for parallelization. We begin by defining a set of equicontrollable states.

Definition 5. $s_1, s_2 \in \Sigma$ *are* equicontrollable *if and only if* $s_1 \in \text{WinSet}(s_2)$ *and* $s_2 \in \text{WinSet}(s_1)$.

In other words, two states are said to be equicontrollable if bidirectional reachability holds.

Parameterized Reachability Games To allow for decomposition in an efficient manner, we consider reachability games that are parameterized in a manner similar to that in [3]. However, in contrast to [3], we allow for liveness properties in addition to assuming safety constraints for the environment.

Let $\mathcal{P}_{\text{AP}}$ be a set of atomic propositions introduced such that $|\mathcal{P}_{\text{AP}}| = |\text{AP}|$ and $\mathcal{P}_{\text{AP}} \cap \text{AP} = \emptyset$. Define a bijective function $f_{\text{param}} : \text{AP} \to \mathcal{P}_{\Sigma}$. Consider some subset of $T \subseteq \text{AP}$ over which we want to parameterize the reachability game. For the set $T \subseteq \text{AP}$, define $\mathcal{P}_T := \{t : \exists x \in T,\ t = f_{\text{param}}(x)\}$. Define an augmented set of variables $\text{AP}^{\mathcal{P}_T} := \mathcal{P}_T \cup \text{AP}$.

We now assume the new auxiliary variables introduced are controlled by the agent, i.e., $\mathrm{AP}_{\mathrm{a}}^{\mathcal{P}_T} = \mathrm{AP}_{\mathrm{a}} \cup \mathcal{P}_T$ and $\mathrm{AP}_{\mathrm{e}}^{\mathcal{P}_T} = \mathrm{AP}_{\mathrm{e}}$. Define the new transition rule for the agent

$$\bar{\rho}^{\mathrm{a}} = \rho^{\mathrm{a}} \wedge \bigwedge_{p \in \mathcal{P}_T} (p \leftrightarrow \bigcirc p).$$

The parametric propositions introduced are constrained to stay fixed during execution, in addition to the original constraints on the agent's behavior. The transition rules for the environment stay unaltered.

Consider a reachability game with the winning condition:

$$\psi_f := \varphi^{\mathrm{e}} \rightarrow \Box \bar{\rho}^{\mathrm{a}} \wedge \Diamond \bigwedge_{t \in T} \left(t \leftrightarrow f_{\mathrm{param}}(t)\right). \tag{2.11}$$

Solving for the set of winning states for this reachability game returns the set of admissible parameters and the corresponding states in Σ that, in combination with the admitted parameters, are winning for condition (2.11). If $(s, r) \in \Sigma \times \mathcal{P}(\mathcal{P}_T)$ and $r \in \mathcal{P}(\mathcal{P}_T)$ are winning for condition (2.11), then what this implies is that starting from s, the controlled agent can force the execution to transition into a state satisfying $f_{\mathrm{param}}^{-1}(r)$. Note that $f_{\mathrm{param}}^{-1}(r) \subseteq T$ may only partially constrain the propositions in AP. Hence, $f_{\mathrm{param}}^{-1}(r)$ can be satisfied by multiple states in Σ.

Remark 6. The set of winning states for condition (2.11) can be computed in $O(|\Sigma|^2)$ symbolic steps in the worst case.

This follows as a direct consequence of Lemma 9 from [76]. From the μ-calculus formula in [76], we note that the non-parameterized reachability game takes worst-case $O(|\Sigma|^2)$ symbolic steps. For a given valuation of the parameters (r), the parameterized reachability game corresponds to solving a reachability game with $S^* = \{s : s|_T = f_{\mathrm{param}}^{-1}(r)\}$ as the set of states to be reached. Thus, the symbolic set operations can be seen as operating on copies of the same transition system for different valuations of the parametric propositions in parallel [3]. Since the parameters stay fixed during execution, adding the parameters does not result in an increase in the number of (worst-case) symbolic steps needed for solving a non-parameterized reachability game. However, the symbolic steps themselves are more

expensive because of the added parameters.

Example 1. Consider a system with AP = $\{a, b, c\}$ and $\Sigma = \mathcal{P}(\text{AP})$. $\mathcal{P}_{\text{AP}} = \{p_a, p_b, p_c\}$ and for $r \in \text{AP}$, f_{param} is defined as $f_{\text{param}}(r) = p_r$. We seek to parameterize the *WinSet* computation over $T = \{b, c\}$, therefore we set $\mathcal{P}_T = \{p_b, p_c\}$. The state $\{a, b\}$ is in *WinSet* of the states satisfying $(b \wedge c)$ if and only if $\{b, p_b, p_c\}$ is a winning state for the condition in equation (2.11). This implies that with $\{b\}$ as the initial state, the agent can force the execution to a state satisfying $(b \wedge c)$.

Partitioning a Set into Equicontrollable Sets

Problem Statement. *Partition the set of states in* Σ *satisfying the Boolean formula* ξ *over propositions in* AP *into equicontrollable classes over the set of propositions* $\mathcal{X}$.

By partitioning over $\mathcal{X}$, we imply that for any $x_1, x_2 \subseteq \mathcal{X}$, the sets of states $\mathcal{S}_1 = \{s | s \in \Sigma, s|_{\mathcal{X}} = x_1\}$ and $\mathcal{S}_2 = \{s | s \in \Sigma, s|_{\mathcal{X}} = x_2\}$ are in the same equicontrollable class if and only if from every $s \in \mathcal{S}_1$, the agent can force the execution into $\mathcal{S}_2$ and vice versa. We slightly abuse the definition of an equicontrollable class by allowing for the states associated with $x \subseteq \mathcal{X}$ to be in the same class even though every pair of states $s_1, s_2 \in \{s | s \in \Sigma, s|_{\mathcal{X}} = x\}$ may not be equicontrollable. This is done since we are interested in partitioning over $\mathcal{X}$.

From here on, we restrict $\mathcal{X}$ to be the set of supporting propositions for the formula ξ, where $[\![\xi]\!]$ is the set of states to be separated into equicontrollable classes. For specific tasks, domain knowledge could guide the selection of the set of the propositions $\mathcal{X}$ to be different from the support variables for the formula ξ.

Algorithm 1 formally describes the procedure for solving the partitioning problem. Let $\mathcal{X}$ be the set of propositions over which we want to separate the equicontrollable classes. First, consider the following formula:

$$\varphi^{\text{e}} \rightarrow \square \rho^{\text{a}} \wedge \diamond \left(\xi \wedge \bigwedge_{t \in \mathcal{X}} \left(t \leftrightarrow f_{\text{param}}(t) \right) \right). \tag{2.12}$$

Solving for the winning states of the above parameterized reachability game gives us a set of states of the form (s, r) with $s \in \Sigma$ and $r \subseteq \mathcal{P}_{\mathcal{X}}$. By construction, these states have the

Algorithm 1: Separating into equicontrollable Classes

Input :

- Environmental behavior φ^{e}, System safety/transition rules ρ^{a}.
- Specification ξ representing the set of states to be separated ($[\![\xi]\!]$).
- BDD ρ^{reach} representing the set of reachable states for the system.
- Set of propositions $\mathcal{X} \subseteq \mathrm{AP}$ over which the states must be partitioned and the map f_{param}.

Output :

- Equicontrollable classes $\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_k$ s.t. $\alpha_i \cap \alpha_j = \emptyset$ for $i \neq j$, $\bigcup_{l=1}^{k} \alpha_i = [\![\xi]\!]$.

Define $\varphi_{\xi}^{\mathrm{param}} := \varphi^{\mathrm{e}} \rightarrow \Box\rho^{\mathrm{a}} \wedge \Diamond \left(\xi \wedge \bigwedge_{t \in \mathcal{X}} \left(t \leftrightarrow f_{\mathrm{param}}(t) \right) \right)$
Compute winning states ($W_{\varphi_{\xi}^{\mathrm{param}}}$) for $\varphi_{\xi}^{\mathrm{param}}$
Equicontrollable Classes = $\emptyset$
for $x \subseteq \mathcal{X}$ **do**
    $t_1 = f_{\mathrm{param}}^{-1}(x)$; $EquivFlag = 0$
    for $p \in$ *Equicontrollable Classes* **do**
        $t_2 = f_{\mathrm{param}}^{-1}(p)$
        if $\left(\exists s.s|_{\mathcal{X}} = x \wedge (s, t_2) \in W_{\varphi_{\xi}^{\mathrm{param}}} \wedge \exists s.s|_{\mathcal{X}} = p \wedge (p, t_1) \in W_{\varphi_{\xi}^{\mathrm{param}}} \right)$ **then**
            $EquivFlag = 1$
        end
    end
    if $EquivFlag = 0$ *and* $(\exists s \in \Sigma.s \models \rho^{\mathrm{reach}} \wedge s|_{\mathcal{X}} = x)$ **then**
        Equicontrollable Classes = Equicontrollable Classes $\cup \{s : s \in \Sigma, s|_{\mathcal{X}} = x\}$
    end
end
return Equicontrollable Classes

property that $f^{-1}_{\text{param}}(r) \models \xi$ and from the state s, the controlled agent can force the execution to reach the set of states $\{s \in \Sigma : s|_{\mathcal{X}} = f^{-1}_{\text{param}}(r)\}$, or block the environment from satisfying the liveness assumptions. In the latter case, the game is trivially won; we will ignore this case hence forth, and it does not affect any of the arguments that follow.

Following this construction, we iterate through the values for the parameters (recall that the parametric propositions have a direct correspondence with the variables in $\mathcal{X}$) and split them into equicontrollable classes as in Algorithm 3. This requires us to perform at most $O(k|\Sigma_{\mathcal{X}}|)$ evaluations once we have computed the winning set for (2.12), where k is the number of classes. Furthermore, while iterating over sets of states, we can eliminate spurious classes by ignoring those sets that have no elements in common with Σ_{reach}.

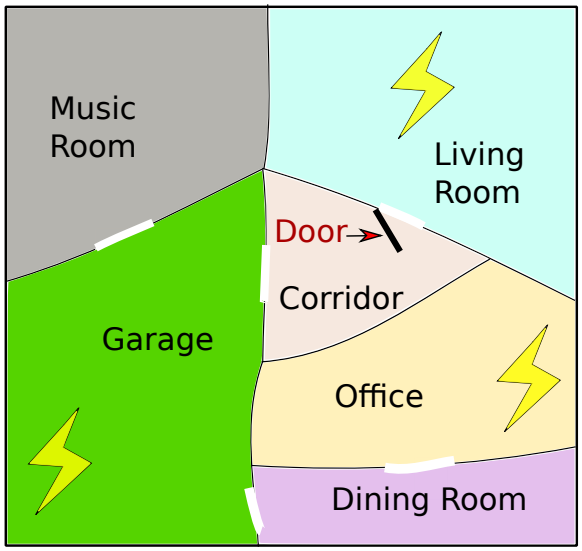


Figure 2.2: Example Workspace

Example 2. For the workspace in Figure 2.2 (similar to that in Figure 2.1, without Door 1), we want to partition the set of states where the robot is in a room with a charging station (lightning sign) into equivalence classes. The specification ξ has the form:

$$\xi = \text{Office} \ \vee \ \text{Living Room} \ \vee \text{Garage}.$$

This specification is satisfied when the robot is in a room that has a charging station. And we set $\mathcal{X} := \{\text{Office, Living Room, Garage}\}$, the supporting variables for ξ.

Suppose $\varphi^e = \texttt{True}$, i.e., the behavior of the door is unconstrained. This yields that Garage, Office are in the same equicontrollable class while the Living Room is in a different class. When we assume that the door opens infinitely often ($\varphi^e = \square\diamond$Door Open) as a model for the environments behavior, the states corresponding to Garage, Office and Living Room are in the same equicontrollable class.

Synthesizing a Composite Controller Next, we will describe an approach to build a transition system and a specification such that the winning strategy for this system can be used to compose the sub-strategies that are synthesized and stored in parallel to find a strategy winning against a GR(1) specification.

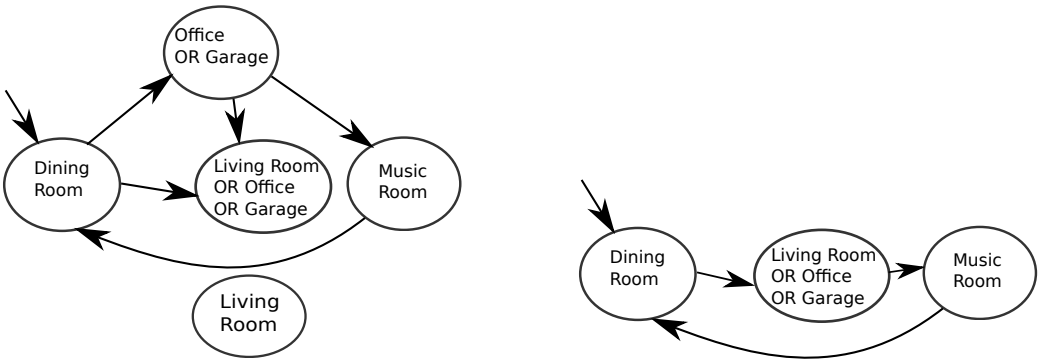
Example 3. For the workspace from Example 2, consider a synthesis problem where the robot has to patrol the dining room and the music room infinitely often, while making sure to visit a room with a charging station infinitely often and the robot is initially in the dining room. The liveness guarantees to be satisfied are:

$$\square\diamond \text{ Dining Room}\,, \qquad \square\diamond (\text{Office} \vee \text{Garage} \vee \text{Living Room})\,, \qquad \square\diamond \text{Music Room}.$$

We fix the ordering of liveness guarantees (Dining Room, Office$\vee$Garage$\vee$Living Room, Music Room) and build a new transition system as shown in Figure 2.3. For each liveness guarantee, there is state in the transition system corresponding to a subset of equicontrollable classes arising from decomposition of the states satisfying the liveness guarantee. We add transitions between these states if the predecessor is in the *WinSet* of the successor (or can block the environment from satisfying the liveness assumptions).

If the environment's behavior is modeled as $\varphi^e = \texttt{True}$, we get the abstracted supervisory transition system shown in Figure 2.3a. For the liveness guarantee Office $\vee$ Garage $\vee$ Living Room, the classes are $\{\text{Office} \vee \text{Garage}, \text{Living Room}\}$. The transition system in Figure 2.3a has states corresponding to all non-empty subsets of these classes. If we assume that the door infinitely often opens, the supervisory transition system is that shown in Figure 2.3b. Both transition systems have a cycle that can be used to compose the sub-strategies.

The construction of the transition system and the composing of the sub-strategies are



(a) Door need not open infinitely often (b) Door opens infinitely often

Figure 2.3: Supervisory transition system for different environment behavior

formally described in the next section.

2.3.1 Composite Controller for Assembling Sub-Strategies

Consider the GR(1) formula in equation (2.2). The states corresponding to the liveness guarantees for the agent (ψ_i^{a}) are partitioned into equicontrollable classes as described above. For each $i \in \{1, 2, \ldots, n\}$, let k_i be the number of classes $[\![\psi_i^{\mathrm{a}}]\!]$ has been partitioned into. Let $\Lambda_i = \{\alpha_{i,1}, \alpha_{i,2}, \ldots, \alpha_{i,k_i}\}$ be the set of classes associated with ψ_i^{a}. Define Ω_i to be set of all subsets of Λ_i except the empty set ($\emptyset$). Note that $|\Omega_i| = 2^{k_i} - 1$. Without loss of generality, we fix some ordering of the elements in Ω_i such that $\Omega_i = \{\Omega_{i,1}, \Omega_{i,2}, \ldots, \Omega_{i,2^{k_j}-1}\}$. Denote by $A_{\Omega_{i,j}}$ the *WinSet* for $\Omega_{i,j}$, i.e., $A_{\Omega_{i,j}} = \mathit{WinSet}(\Omega_{i,j})$. For the case in Example 3, where the door is not assumed to open infinitely often, the classes corresponding the specification where the robot has to infinitely often visit a room with a charging station are

$$\Lambda_2 = \{\text{Office} \vee \text{Garage},\ \text{Living Room}\},$$

and

$$\Omega_2 = \Big\{ \{\text{Office} \vee \text{Garage}\}, \{\ \text{Living Room}\}, \{\text{Office} \vee \text{Garage},\ \text{Living Room}\} \Big\}.$$

Note that if $|\Omega_{i,j}| = 1$ we can use the parametric *WinSet* computed for decomposing $[\![\psi_i^{\mathrm{a}}]\!]$ into equicontrollable classes to obtain the *WinSet* for $\Omega_{i,j}$ by setting values to the

parametric propositions appropriately. For $\Omega_{i,j}$ with $|\Omega_{i,j}| > 1$, we compute $WinSet(A_{\Omega_{i,j}})$ by solving a reachability game with $\Omega_{i,j}$ as the goal to be reached. As these computations are independent, they can be performed in parallel.

Following this setup, the hierarchical game is constructed as follows. The set of atomic propositions are

$$\overline{\mathrm{AP}} = \{\rho_{i,j_i} : i \in \{1, 2, \ldots, n\}, j_i \in \{1, 2, \ldots, 2^{k_i} - 1\}\}.$$

The transition rule is specified as

$$\rho^{\text{compositional}} = \underline{\bigvee_{v \in \overline{\mathrm{AP}}}} v \quad \wedge \underbrace{\bigwedge}_{\substack{i \in \{1,2,\ldots,n\} \\ j \in \{1,2,\ldots,2^{k_i}-1\}}} \left(\rho_{i,j} \rightarrow \bigcirc \underbrace{\bigvee}_{\substack{\Omega_{i,j} \subseteq A_{\Omega_{k,l}} \\ k \in \{i, i \oplus 1\} \\ l \in \{1,2,\ldots,2^{k_i}-1\}}} \rho_{k,l} \right). \tag{2.13}$$

Here $\underline{\bigvee}$ is the *XOR* operator. In equation (2.13), we allow for a transitions between the states $\{\rho_{i,j}\}$ and $\{\rho_{k,l}\}$ only if the current $\Omega_{i,j}$ is in the *WinSet* for the $\Omega_{k,l}$ corresponding to the successor state. For the transition system in Figure 2.3a, the Dining Room is in the *WinSet* of Living Room $\vee$ Office $\vee$ Garage but Living Room $\vee$ Office $\vee$ Garage is not in the *WinSet* of Music Room. The transition relations reflect the same. Similarly, transition relations are constructed between the other states.

Note that we restrict $k \in \{i, i \oplus 1\}$, ensuring that only those transitions that either stay in the same liveness guarantee or lead to the next liveness guarantee are chosen. This way we do not cycle back to a liveness guarantee that was visited earlier in the current cycle. This makes the transition rules sparse, keeping the BDD small, thereby reducing the time required for synthesizing the composite controller.

The liveness guarantees ensure that infinitely often for each $i \in \{1, 2, \ldots, n\}$, $\rho_{i,j}$ for

some j is satisfied. The liveness guarantees can be formally written as

$$\psi_i^{\text{compositional}} := \bigvee_{j \in \{1,2,\ldots,2^{k_i}-1\}} \rho_{i,j}. \tag{2.14}$$

This ensures that at least one of the classes corresponding to a liveness guarantee is visited in each cycle through the liveness guarantees. For a particular i, satisfying $\psi_i^{\text{compositional}}$ is equivalent to satisfying ψ_i^{a} in the original system.

Note that here there is no environment here and we only need to search for a cycle passing through all the liveness guarantees for a given set of initial states. Consider some $i \in \{1, 2, \ldots, n\}$. The set of valid initial states are the nodes corresponding to the elements in Ω_i for which $[\![\theta^{\text{a}} \wedge \theta^{\text{e}}]\!]$ lies in their *WinSet*. The initial condition can be written as (for some i)

$$\theta^{\text{compositional}} := \bigvee_{\theta \subseteq A_{\Omega_{i,j}}} \rho_{i,j} \ . \tag{2.15}$$

Composing the specifications above, we need to find a controller for the condition

$$\theta^{\text{compositional}} \wedge \Box\rho^{\text{compositional}} \wedge \bigwedge_{i=1}^{n} \Box\Diamond\psi_i^{\text{compositional}} \ . \tag{2.16}$$

Finding a winning strategy $f^{\text{compositional}} : \overline{\text{AP}} \times M^{\text{sup}} \to \overline{\text{AP}} \times M^{\text{sup}}$ (where M^{sup} is a set of memory values) for the above specification gives the compositional controller for composing the strategies for the reachability games.

2.3.2 Composing the Sub-Strategies

Define $f_{i,l}^{k,l} : \Sigma_{\text{e}} \times \Sigma \times M_{i,l}^{k,l} \to \Sigma_{\text{a}} \times M_{i,l}^{k,l}$ to be the strategy that takes the agent from a state in $\Omega_{i,l}$ to $\Omega_{k,l}$. Let $\underline{m}_{i,j}^{k,l}$ be the initial memory value for $M_{i,l}^{k,l}$. Without loss of generality, assume $M_{i_1,l_1}^{k_1,l_1} \cap M_{i_2,l_2}^{k_2,l_2} = \emptyset$ when $(i_1, j_1, k_1, l_1) \neq (i_2, j_2, k_2, l_2)$.

Define $k_{\max} := \max\{k_i : i \in \{1, 2, \ldots, n\}\}$, i.e., $k_{\max}$ is the size of the largest number of equicontrollable classes for any of the liveness classes. Let $\overline{M} := M^{\text{sup}} \times \bigcup_{i,j,k,l} M_{i,j}^{k,l}$ and $\xi_{\text{comp}} := \{1, 2, \ldots, N\} \times \{1, 2, \ldots, k_{\max}\} \times \{1, \ldots, N\} \times \{1, \ldots, k_{\max}\}$.

We construct a strategy:

$$f^{\text{compose}} : \Sigma_{\text{e}} \times \Sigma \times \xi_{\text{comp}} \times \overline{M} \to \Sigma_{\text{a}} \times \xi_{\text{comp}} \times \overline{M}$$

that uses $f^{\text{compositional}}$ to compose the strategies for the reachability games ($f^{k,l}_{i,l}$):

$$f^{\text{compose}}(x, s, i, j, k, l, w, w_{\text{sup}}) = (y, i', j', k', l', w', w'_{\text{sup}}), \tag{2.17}$$

where if $s \notin \Omega_{k,l}$, then

$$\begin{aligned} (y, w') &= f^{k,l}_{i,j}(x, s, w), \\ (i', j', k', l') &= (i, j, k, l), \\ w'_{\text{sup}} &= w_{\text{sup}}, \end{aligned} \tag{2.18}$$

and if $s \in \Omega_{k,l}$, then

$$\begin{aligned} (y, w') &= f^{k,l}_{i,j}(x, s, \underline{m}^{k,l}_{i,j}), \\ (\{\rho_{k',l'}\}, w'_{\text{sup}}) &= f^{\text{compositional}}(\{\rho_{k,l}\}, w_{\text{sup}}), \\ (i', j') &= (k, l). \end{aligned} \tag{2.19}$$

In equation (2.18), while we are moving towards $\Omega_{k,l}$, the values are updated according to the strategy $f^{k,l}_{l,j}$. Once we reach $\Omega_{k,l}$ (equation (2.19)), the next goal is updated according to $f^{\text{compositional}}$ and we continue towards the next goal, switching goals again once the next goal is reached.

Theorem 7. *Strategy f^{compose} is sound. Solving equation* (2.16) *takes in the worst-case $O((2^{k_{max}})^2 n^3)$ symbolic steps.*

Proof. By construction, a winning strategy in the original system was computed corresponding to every transition in the abstracted system, i.e., the agent can either force the execution to the next liveness guarantee or block the environment from satisfying the assumption on its behavior. A winning strategy for the abstracted system finds an execution that cycles through the liveness guarantees. Cycling through the liveness guarantees in

the abstracted system corresponds to cycling through liveness guarantees in the original system (or being able to block the environment from satisfying its assumptions). Hence, composing the strategies from the reachability games in accordance with the composite controller ensures satisfaction of the original GR(1) formula.

The specification resulting in equation (2.16) is a GR(1) formula without an environment, i.e., it is not reactive, hence the innermost fixpoint associated with blocking the environment from satisfying its assumptions does not add to the number of symbolic steps to be performed. The total number of states is $(n2^{k_{max}})$ and there are n liveness guarantees, resulting in $\mathcal{O}((2^{k_{max}})^2 n^3)$ symbolic steps [48]. □

For applications where the number of liveness guarantees and the number of equicontrollable classes are much smaller than the total number of states, i.e., $n \ll |\Sigma|$ and $k \ll |\Sigma|$, the parallelized approach presented here is well-suited and should result in performance gains in term of computation time. We expect such behavior in multi-agent systems with large state spaces where the agents' dynamics are not closely coupled.

Limitations

There can be potential corner cases where the algorithm presented above is not complete, as we lose certain transitions during abstraction into the supervisory transition system. Besides completeness another limitation of the approach is that if we end up with a large number of equicontrollable classes, the computation of the compositional strategy can become intractable. We provide a counterexample below where the approach fails.

Example 4. To illustrate a case where the approach outlined here fails, we consider the following counter example. Let the set of atomic propositions be $\mathrm{AP} = \{b, c, d\}$ with $\mathrm{AP}_e = \{b, d\}$.

Define the transition rule for the environment as

$$(d \rightarrow \bigcirc c)\,. \tag{2.20}$$

Define the transition rule for the controlled agent as

$$\rho^e = ((\neg c \wedge (b \vee \neg d) \rightarrow \bigcirc \neg (b \vee d))) \tag{2.21}$$

Let the initial condition be $\theta = (c \wedge b)$. Consider the following GR(1) synthesis problem:

$$\theta \wedge \square \rho^e \rightarrow \square \rho^a \wedge \square \diamond b \wedge \square \diamond d. \tag{2.22}$$

The winning states for this problem are $\{(b, c, d), (b, c)\}$. From both of those states the agent can pick d and $\neg b$ to hold at the next state, forcing c to hold two instants into future. When c holds, the agent can pick b satisfying the $\diamond b$ and then, it is allowed to pick d and $\neg b$ at the next instance and so on, the cycle can continue.

However, when we use the hierarchical approach, we do not obtain a cycle between the liveness guarantees. The controlled agent cannot force the execution to satisfy $\diamond d$ from all states b. To see this consider the state $\{b\}$. $\neg (b \vee d)$ has to hold at the next step and if the environment decides to set $\neg c$, $\neg b \vee d$ has to again hold at the next instant and this goes on. Hence, though we have a winning strategy, we are not able to find it in the abstracted system, demonstrating the incompleteness of the approach. However, if the partitioning of $[\![b]\!]$ was parameterized over both b and c, the hierarchical approach would have had a cycle.

2.4 Experiments and Analysis

In this section, we perform experiments to benchmark the performance of the different approaches.

2.4.1 Parallelized Synthesis for Singleton Liveness Guarantees

Here, we study the simpler problem setting of synthesizing strategies for GR(1) specifications with singleton liveness guarantees. For this class of specifications, in Section 2.2 we propose a parallelized approach for synthesis based on decomposition into $n + 1$ independent reachability games. To compare the performance of this parallelized approach with

standard GR(1) synthesis, we consider the problem of coordinated planar reactive robot motion planning on a gridworld. For a given set of cells $\{(a_1^r, a_1^c), (a_2^r, a_2^c), \ldots, (a_n^r, a_n^c)\}$ and $\{(b_1^r, b_1^c), (b_2^r, b_2^c), \ldots, (b_n^r, b_n^c)\}$, the controlled robot has to coordinate with a moving agent such that the robot is in cell (b_i^r, b_i^c) when the agent is in cell (a_i^r, a_i^c). The robot has to complete this coordination task infinitely often. To make sure the problem is feasible, the cells $\{\{(b_1^r, b_1^c), \ldots, (b_n^r, b_n^c)\}\}$ are added as liveness conditions for the agent. The robot's motion constraints allow movement to any of its non-diagonally adjacent cells.

Let Y_r denote the row (horizontal) position of the controlled robot Y_c the column (vertical) position. Similarly, let X_r and X_c denote the row and the column position of the uncontrolled agent. The transition rule for the robot at position $(Y_r, Y_c) = (i, j)$ can be written as

$$\begin{aligned}(Y_r = i \wedge Y_c = j) \rightarrow \Big(& \big(Y_r' = i + 1 \wedge Y_c = j\big) \\ & \vee \big(Y_r' = i \wedge Y_c' = j + 1\big) \\ & \vee (Y_r' = i \wedge Y_c' = j) \\ & \vee (Y_r' = i - 1 \wedge Y_c' = j) \\ & \vee (Y_r' = i \wedge Y_c' = j - 1)\Big),\end{aligned}$$

with the additional constraint that Y_r and Y_c always stay in the bounds of the gridworld, i.e.,

$$0 \leq Y_r \leq r_{\max}, 0 \leq Y_c \leq c_{\max}.$$

The agent's motion is constrained in a similar way. Furthermore, the controlled robot as a part of the safety specification has to avoid collision with the uncontrolled agent, i.e.,

$$\neg(X_r = Y_r \wedge Y_c = X_c),$$

where $\neg$ is the negation operator. Both the controlled robot and the uncontrolled agent have to avoid collisions with the walls (shaded). For example, if the location (w_r, w_c) is shaded, then the safety specification corresponding to avoiding collision with this wall for

the controlled robot is

$$\neg(Y_r = w_r \wedge Y_c = w_c).$$

The liveness assumptions can be specified as

$$\bigwedge_{i=1}^{n} \square\diamond(X_r = b_i^r \wedge X_c = b_i^c).$$

The liveness guarantees are written as

$$\bigwedge_{i=1}^{n} \square\diamond(Y_r = a_i^r \wedge Y_c = a_i^c \wedge X_r = b_i^r \wedge X_c = b_i^c).$$

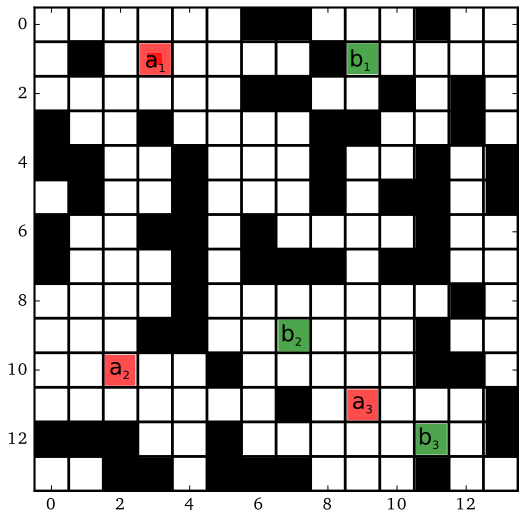


Figure 2.4: Gridworld of size 14×14 with wall density of 0.3. An example task with singleton liveness goals, where the uncontrolled robot has to cycle between b_1, b_2 and b_3 infinitely often. The controlled robot has to be at a_1 when the uncontrolled robot is at b_1, and so on for (a_2, b_2) and (a_3, b_3), while avoiding collision.

Figure 2.4 shows an example gridworld instance. The runtimes for the approach presented here using the decomposed reachability games is compared with those for the solvers `gr1c` [84] and `slugs` [47]. The solvers are accessed using the interfaces in the Temporal Logic Planning Toolbox (`TuLiP`)] [51]. The reachability games for decomposition-based approach are solved using the `rg` module from `gr1c`. The computations were performed on

a 2.40 GHz quad-core machine with 16 GB of RAM.

Gridworld instances with varying number of liveness guarantees and gridsizes are used for benchmarking. Figure 2.5 depicts the mean runtimes from the benchmarking experiments on $t \times t$-sized gridworld instances, with varying t. For each grid size, 50 random problem instances (with a wall density of 10% and 6 liveness guarantees) are created. We see that the decomposition-based approach outperforms GR(1) synthesis (using `slugs` and `gr1c`).

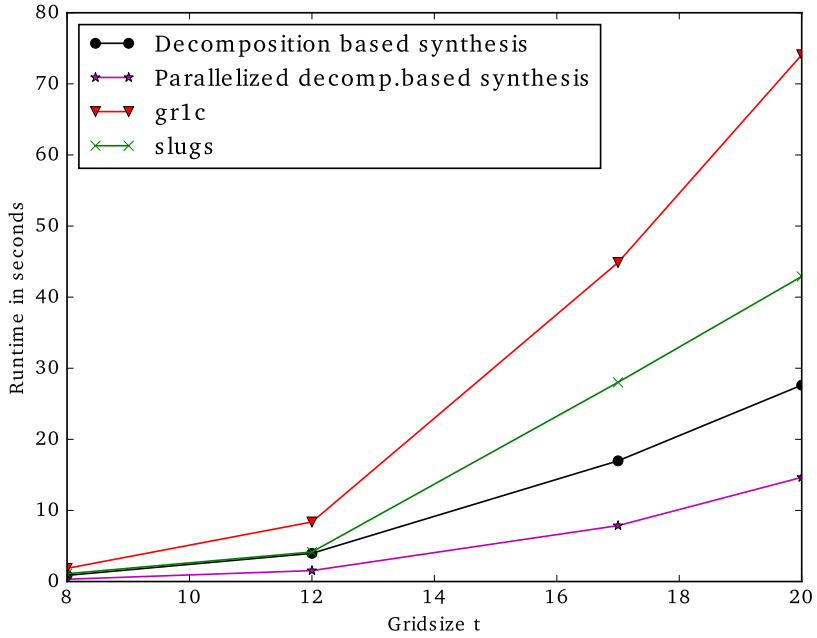


Figure 2.5: Performance on gridworld problems with varying grid size ($t \times t$).

Figure 2.6 depicts the performance for gridworld instances of size 14×14 (wall density 0.3) with the number of liveness constraints changing. Here again we observe similar trends with the decomposition approach outperforming GR(1) synthesis using `slugs` and `gr1c`. When the reachability games are solved in parallel, we observe improved scaling for the decompositio-based approach. The slope for the parallelized decomposition-based synthesis is less than that of decomposition-based synthesis without parallelization. This is because the complexity of each of the reachability games is independent of n, where n is the number of liveness guarantees. Since the $n+1$ reachability games are solved in parallel, the runtime approximately stays constant even with the varying number of liveness guarantees.

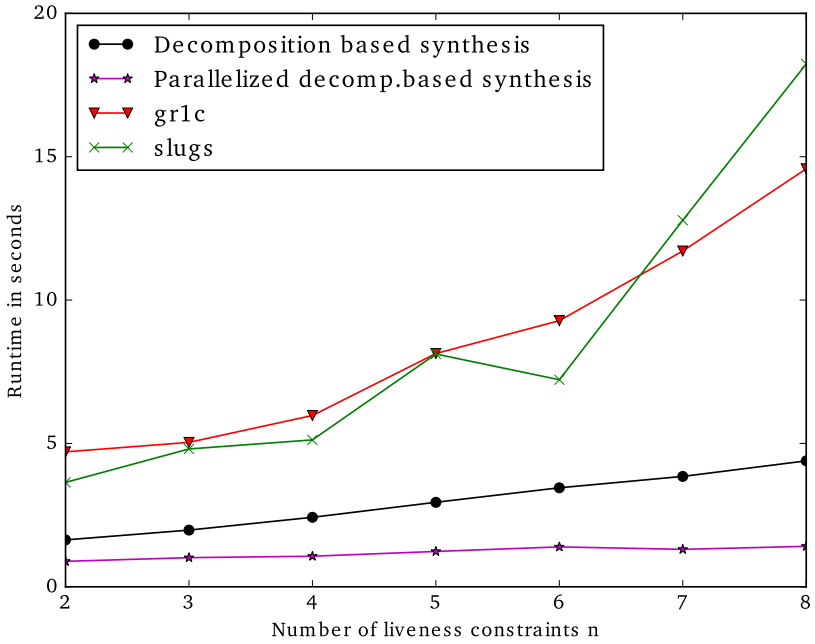


Figure 2.6: Performance on gridworld problems with varying number of liveness constraints.

2.4.2 Parallelized Synthesis via Equicontrollable Classes

Next, we study the hierarchical synthesis approach from Section 2.4.2 that allows for parallelization. We consider a multi-agent robot motion planning problem where the objective is to, for a set of robots, schedule access to critical sections of a given workspace in a safe manner. The environment consists of an uncontrolled adversarial mobile robot that is functioning in the same workspace as the controlled robots and requires access to certain critical sections of the workspace. An example instance is shown in Figure 2.7. Both the controlled robots and the uncontrolled robot have to visit cells shaded with each of the three colors (red and green) infinitely often. The problem instances are parameterized in terms of the size of the workspace, the number of critical sections and the number of controlled robots. The colored cells represent critical sections of the workspace that must be accessed in a mutually exclusive manner and each color represents a critical resource of a type. While the adversarial robot is accessing a critical section, the controlled robots must not access the same critical section. Similarly, no two controlled robots can access a critical section at the same time. The adversarial robot's access to the critical sections is prioritized over the controlled robots. When the adversarial robot attempts to access a critical space,

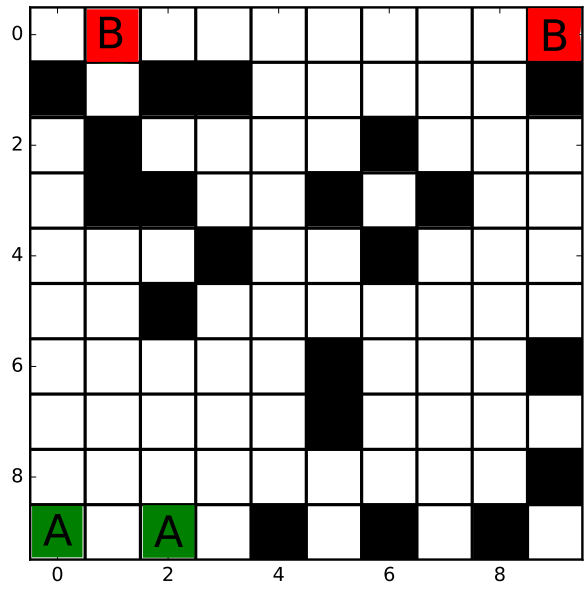
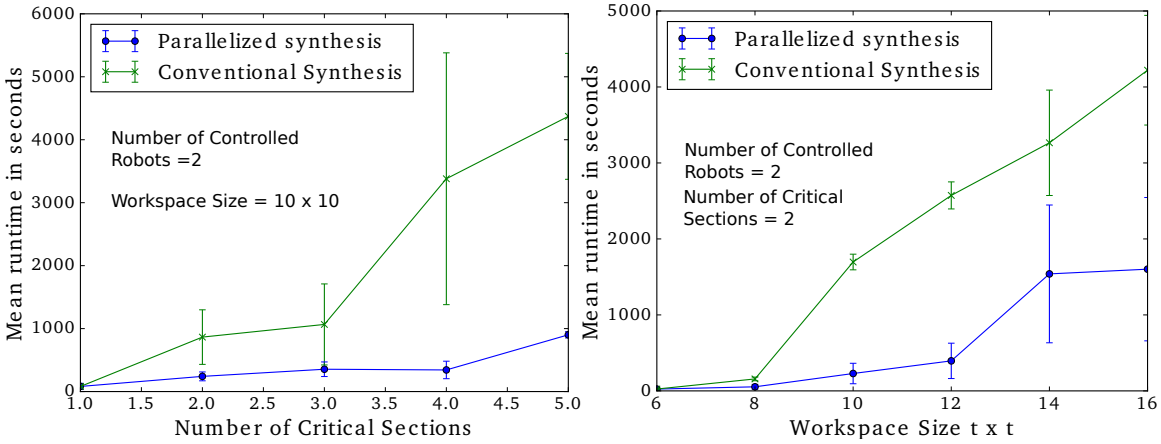


Figure 2.7: Workspace with shaded obstacles (black) and critical sections (green and red). The uncontrolled robot has to cycle between visiting a green cell (marked A), and a red cell (marked B). The controlled robot has to do the same, while allowing the uncontrolled robot priority access to any colored cell. The critical sections (green and red) can only be occupied by one robot at any given time.

the controlled robots as a part of their safety requirement must allow the adversarial robot to gain access by vacating the critical section. The regions shaded black (density=5%) represent static obstacles and both the uncontrolled and controlled robots must avoid the obstacles. The robots are allowed to transition to any of their non-diagonally adjacent cells in a single step. The uncontrolled robot is allowed to pursue a trajectory of its choice and the only assumption on its behavior, in addition to the constraints on its motion, is that the uncontrolled robot will access cells shaded with each of the colors infinitely often.

Figures 2.8a, 2.8b report performance over problem instances of varying size. The mean time over 50 problem instances is reported. For each of these instances the initial positions of the robots, the positions of the obstacles and the critical sections are randomized. The computations were performed on a 32 core AMD Opteron machine at 2.4 GHz with 96 GB of RAM, and we see considerable gains in computation time for the parallelized approach.



(a) Varying Number of Critical Sections

(b) Varying size of the workspace

Figure 2.8: Performance on benchmark experiments. Mean runtimes over 50 randomized problem instances are reported, error bars indicate standard deviation.

2.5 Conclusions and Future Work

A major challenge to the widespread adoption of formal methods is their scalability. As systems get larger and complex, scalable algorithms that can deal with the size and complexity of the systems are necessary. In this regard, we present approaches that allows us to decompose and parallelize the synthesis algorithm for the GR(1) fragment of linear temporal logic. The approaches rely on the construction of a composite strategy that is used to compose local strategies to ensure satisfaction of the GR(1) specification. However, the approach comes with certain drawbacks as outlined earlier. Empirical evidence demonstrating the resulting gains in performance is presented for robot motion planning problems.

Future work would be to explore if similar approaches can be used to synthesize policies that can handle uncertainty, because some local uncertainty can be tolerated without a resynthesis of the entire strategy by applying a local correction. A hierarchical framework as presented here could be used in such settings. Another direction for future work includes exploring the possibility of a symbolic approach for decomposition of sets into equicontrollable classes as opposed to the enumerative approach considered here.

Chapter 3

Learning for Verification

Verification plays a key-role in designing reliable systems, ranging from planning and scheduling for control systems to program analysis. However, verification often suffers from two critical bottlenecks:

- Verification of complex autonomous systems often involves solving hard, difficult to scale constraint-solving problems (e.g., reachability problems for highly nonlinear dynamical systems),
- Verification of autonomous systems requires accurate models of the environments with which the system interacts. However, we often require our autonomous complex to interact with complex environments that are difficult to model (e.g., a autonomous vehicle that has to interact with pedestrians, who are difficult to model).

This chapter proposes approaches to learning abstractions that alleviate some of the concerns on both fronts, and thereby enabling the wider usage of available tools for verification. First, we consider the verification of cyber-physical systems with lookup tables. Lookup tables are an important and irreplaceable element of modern engineering design. Lookup tables are frequently used to approximately model highly nonlinear physical components, which are difficult to model. Others serve as control laws in cases where no traditional control design method delivers the required performance. This widespread use of lookup tables in embedded systems, across industries such as aeronautics and automotive systems, creates a critical obstacle for scalable formal verification. Lookup tables challenge traditional verification techniques because each entry of the lookup table must be treated as a separate

case. If the system under analysis contains a large number of cascaded lookup tables, the number of proof cases grows exponentially, quickly outstripping the ability to deliver timely verification results as part of a product-development cycle. To overcome this bottleneck, we introduce an approach for computing abstractions of lookup tables that are provable overapproximations and have a simple form. These simpler abstractions can then replace the original lookup tables during verification, significantly reducing the complexity of the verification problem. We illustrate the performance of our approach on a cruise control benchmark [124]. This benchmark consists of a controller with a monitor that tries to detect dangerous conditions. This benchmark contains three lookup tables. The simplest lookup table is one-dimensional and the most complex lookup table is three-dimensional. The total number of combinations of lookup table outputs is $77,409,024$.

We then generalize the insights from verifying lookup tables to propose a more general learning-based approach for abstracting hard constraint problems to find simple (provable) relaxations or restrictions of the original problem. To this end, we introduce an approach (based on [33]) to decompose a set of nonlinear arithmetic constraints and learn simpler relaxations and restrictions for the decompositions. To learn these simplifications, we compute a large number of satisfying and falsifying instances for each subset of constraints using a sampling procedure described later and use these as training data for the learning procedure. For a given set of samples, we use a *semi-soft* support vector machine (SVM) to learn asymmetric classifiers as candidate *antecedents* and *consequents*. The semi-soft SVM is embedded into a refinement loop where a reduced constraint problem is solved to ensure that the candidate antecedent and consequent are indeed an antecedent and consequent. The simpler learned antecedents can be used to find a satisfying instance, and the learned consequents can be used to demonstrate that no satisfying instance exists. Other attempts at using learning to find interpolants in domains such as program analysis and safety analysis for hybrid systems [103] do not address the problem of scalability. We evaluate our technique on four benchmark constraint sets corresponding to the reachability analysis for a toy-car, finding valid encoder expressions for FPGA design, random instances of polynomial constraints and Hong's problem in the first quadrant. Our experiments show that this technique provides an improvement in the scale of problems that can be handled

by a constraint solver.

In the latter part of the chapter, to overcome the challenge from accurate models being difficult to obtain, we present a data-driven approach that allows for a system to be evaluated for specification conformance without an accurate model of the environment. Our approach involves learning a conservative reactive bound of the environment's behavior using data and specification of the system's desired behavior. The approach begins by learning a conservative reactive bound on the environment's actions that captures its possible behaviors with high probability. This bound is then used to assist verification, and if the verification fails under this bound, the algorithm returns counterexamples to show how failure occurs and then uses these to refine the bound. We demonstrate the applicability of the approach through two case studies: i) verifying controllers for a toy multi-robot system, and ii) verifying an instance of human-robot interaction during a lane-change maneuver given real-world human-driving data.

Constraint Solving: Related Work Several approaches have been proposed to improve performance for discrete constraint satisfiability problems using decompositions, including [8, 31]. [52] a problem-decomposition approach where the original nonconvex optimization problem is decomposed into subproblems that are approximately independent. In contrast, our approach first decomposes the problem and then uses a learning procedure to find simpler abstractions of the problem that are more tractable to solve. The main problem with finding these *interpolants* – which serve as simplifications through syntactic manipulations [27] – is that they are typically slower and difficult to scale.

Data-driven Verification: Related Work Recently, there has been an increased interest in data-driven verification for cyber-physical systems [49, 50, 79]. In [11], a data-driven automated approach is proposed to identify non-converging behaviors in black-box control systems. In [63], the authors propose an approach based on Bayesian inference and reachability analysis for verifying the behavior of systems. However, the approach does not decompose the system into the uncontrolled environment and the controller, and is limited to the model class of linear time-invariant systems. In contrast, our approach lever-

ages known policies for the controlled agent to enable verification of their behavior with complex environments. A closely related direction of work is on mining specifications [72, 116, 117] from data. The mined specifications are often used as task specifications, as opposed to being used to verify a given controller. For the case of human-robot interaction, treating it as a multi-agent task and leveraging the influence of the autonomous agent on the (uncontrolled) human has been considered before [81, 113]. In [100], the authors propose an approach that learns a reward function to model the behavior of the uncontrolled agent and then leverages this reward function to plan for the autonomous agent. This allows the planner to consider the influence of the autonomous agent on the uncontrolled agent. Here the authors incorporate the environments behavior into the planning phase, while in contrast, we leverage the controlled agent's policy and the desired safety specification for the system to characterize the environment's behavior to facilitate verification. [108] demonstrates the benefits of learning the intent of the uncontrolled agent prior to the physical event, and leverages this learned intent for seamless collaboration.

This chapter is structured as follows. First, we begin by providing the necessary background on constraint solving, before describing our approach for learning abstractions for constraint solving instances. Then, we evaluate our technique on four benchmark constraint sets corresponding to the reachability analysis for a toy-car, finding valid encoder expressions for FPGA design, random instances of polynomial constraints and Hong's problem in the first quadrant.

Next, we consider the problem of data-driven verification for autonomous systems. We begin by providing the necessary background regarding Signal Temporal Logic (STL), an extension of LTL to real-valued signals, and random convex programs. Subsequently, we describe the approach for learning abstractions that accurately characterize the environment's behavior with high probability and can be used for verifying desired properties.

3.1 Lookup Tables and Constraint Solving: Background

3.1.1 Lookup Tables

Informally, a lookup table is a function defined by a table of input and output values. A lookup table maps certain points of its input space, called *breakpoints*, to values prescribed by a given table, such as the one shown in Table 3.1. Note that despite the tabular structure, Table 3.1 represents an n-dimensional lookup table, *not* a two-dimensional one. The output of the function for values that do not appear in the table are computed by some given interpolation function if they are contained in the range of the breakpoints, and by some extrapolation function otherwise.

$x_1^{(1)}$	$\dots$	$x_i^{(1)}$	$\dots$	$x_n^{(1)}$	$y^{(1)}$
$\vdots$		$\vdots$		$\vdots$	$\vdots$
$x_1^{(j)}$	$\dots$	$x_i^{(j)}$	$\dots$	$x_n^{(j)}$	$y^{(j)}$
$\vdots$		$\vdots$		$\vdots$	$\vdots$
$x_1^{(m)}$	$\dots$	$x_i^{(m)}$	$\dots$	$x_n^{(m)}$	$y^{(m)}$

Table 3.1: Lookup table with n inputs and m breakpoints

Formally, an n-dimensional lookup table with m breakpoints is a function $\lambda : \mathbb{R}^n \to \mathbb{R}$, such that

1. for each breakpoint $(x^{(k)}, y^{(k)})$ ($k \in \{1, \dots, m\}$) that appears in the table, $\lambda(x^{(k)}) = y^{(k)}$, and

2. for every point $x \in \mathbb{R}^n$ that does not appear in the table,

   (a) if each component x_i is contained in the range of the lookup table, i.e., $\min_k(x_i^{(k)}) \leq x_i \leq \max_k(x_i^{(k)})$ for each $i \in \{1, \dots, n\}$, then $\lambda(x)$ is given by some interpolation function `interp`.

   (b) otherwise, $\lambda(x)$ is given by some extrapolation scheme `extrap`.

Our approach is general, and can be applied to any interpolation and extrapolation functions. However, in our case study, we will interpolate the lookup table by the multilinear

interpolation formula described in [120]. For n dimensions, we will use the notation

$$\texttt{multiLinInterp}_n((x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), x)$$

to mean the n-dimensional interpolation function between points $(x^{(1)}, y^{(1)})$ and $(x^{(2)}, y^{(2)})$, evaluated at x. For simplicity, we will not extrapolate the lookup tables in our case study and simply assume that the range of interest is restricted to the range of the lookup tables.

3.1.2 Lookup Tables as Logical Formulas

Our technique relies on the ability to encode the system and its specification into first-order logic. An n-dimensional, m-breakpoint lookup table can be encoded as a first-order logical formula as follows. Consider, without loss of generality, a two-dimensional lookup table with m breakpoints. The k-th breakpoint can be encoded by the following logical formula

$$\begin{aligned} b_k \equiv & x_1^{(k)} \leq x_1 \leq x_1^{(k+1)} \ \wedge \ x_2^{(k)} \leq x_2 \leq x_2^{(k+1)} \rightarrow \\ & y = \texttt{multiLinInterp}_2((x^{(k)}, y^{(k)}), (x^{(k+1)}, y^{(k+1)}), x), \end{aligned}$$

where $k = 1, \ldots, m-1$. The vector x is the input of the lookup table, and x_1 and x_2 are its components. The function $\texttt{multiLinInterp}_2$ is bilinear interpolation. Similar expressions can be derived for lookup tables of other dimensions. The overall lookup table can be expressed by the conjunction of the logical formulas for the breakpoints.

Satisfiability Modulo Theories Suppose a set of constraints $\{A_1(x) \ldots, A_m(x)\}$ is given, where x represents a vector of variables. The real-valued constraint-solving problem, also called *satisfiability modulo the theory of the reals*, is to compute a real-valued vector $\hat{x}$ that simultaneously satisfies all of the constraints A_i, or to prove that no such vector exists. If a satisfying instance $\hat{x}$ is found, then we say that the constraint set is *satisfiable*, and otherwise, we say it is *unsatisfiable*.

Antecedents and Consequents Let $A(x)$ be a logical formula with vector of free variables x. We say that a logical formula $C(x)$ is an *antecedent* of $A(x)$ if $\forall x \,.\, C(x) \implies A(x)$. This means that the values of x that satisfy C are a subset of the values of x that satisfy A, so we will also call $C(x)$ an *underapproximator* of $A(x)$. Conversely, we say that a formula $D(x)$ is a *consequent* of $A(x)$ if $\forall x \,.\, A(x) \implies D(x)$. In this case, all values of x that satisfy $A(x)$ also satisfy $D(x)$, so we say that $D(x)$ is an *overapproximator* of $A(x)$. If $\{A_1, \ldots, A_k\}$ is a set of constraints, we say that $C(x)$ is an antecedent for the set of constraints if it is an antecedent for the conjunction, i.e.,

$$\forall x \,.\, C(x) \implies A_1 \wedge \cdots \wedge A_k. \tag{3.1}$$

The consequent of a set of constraints is similarly defined to be a logical consequence of the conjunction of the constraints.

3.2 Lookup Table Abstraction: Problem Statement

We consider the problem of proving input-output properties of cyber-physical control systems with lookup tables. We assume the control system has been translated to a set of logical constraints $\Sigma(x)$, not including any lookup tables, where x is the vector of *all* variables that occur in the system, including inputs, outputs, and intermediate assignment variables. We handle the lookup tables separately, and assume that each lookup table, indexed by i has been encoded as the first-order logic formula $L_i(x)$. Similarly, we assume that the specification is given as a first-order formula $S(x)$. Then, the problem is to determine whether there exists a value of the variables x that:

1. satisfies the model constraints $\Sigma(x)$, i.e., the values are related to each other according to the structure of the model;

2. satisfies each L_i, i.e., the values are related to each other in a way that satisfies the mapping produced by the lookup tables; and

3. does not satisfy the specification $S(x)$, i.e., it is an erroneous condition.

To check for the existence of this kind of erroneous condition, we can use an SMT solver to check the satisfiability of the following logical formula, assuming the number of lookup tables in the model is N:

$$\left(\bigwedge_{i=1}^{N} L_i(x)\right) \wedge \Sigma(x) \wedge \neg S(x).$$

This logical formula states that values for the vector of variables x must satisfy each lookup table L_i as well as the model constraints $\Sigma(x)$. In addition, the value x should *falsify* the safety condition $S(x)$. If no such value exists, then the system is guaranteed to be defect-free.

The key obstacle to directly checking this condition is that the lookup table formulas L_i are complex, and combining several complex lookup tables renders the verification problem intractable. Further compounding this problem, each entry of the lookup table is encoded as an implication, which induces a case analysis: each range on the left-hand side of the implication is a case, and the right-hand side of the implication is the value of the table at that case. If we assume for simplicity that all lookup tables have m cases, and that there are k lookup tables in a model, hence the total number of cases is m^k. This exponential explosion in cases forbids a naive analysis.

Our approach is to generate a simple overapproximating function to replace the complex formula L_i with the abstraction A_i by using the lookup table data as training data to learn parameters in an abstraction template. As a result, the logical formula will be simplified, but the abstraction loses information. To address this, we provide a falsification heuristic that can help to find true counterexamples when the verification does not succeed.

3.3 Lookup Table Abstraction: Approach

Our approach to improve scalability is to abstract the lookup tables by *functional intervals*. A functional interval is a function that for each argument $x \in \mathbb{R}^n$ returns a (closed) interval over R, $A(x) = [a(x), b(x)]$ where $a(x)$ is the lower bounding envelope around the lookup table and $b(x)$ is the upper bounding envelope. We say that a functional interval $A(x)$ *abstracts* a lookup table $L(x)$ over a set $S \subseteq \mathbb{R}^n$ if for every $x \in S$, $L(x) \in A(x)$.

A functional interval abstraction is an *overapproximation* of a lookup table, in the sense that a property that holds for all values in the interval $A(x)$ must also hold for $L(x)$, but not *vice versa*. The abstraction loses precision, but provides a simplification if the functions $a(x)$ and $b(x)$ have a sufficiently simple structure.

As a result, a procedure to compute a functional interval abstraction must balance between two conflicting requirements. On the one hand, it should be as precise as possible, by keeping the size of the interval small for every x, but it must also have a simple arithmetic structure, preferably consisting of linear or low-order polynomial terms, so that proving that the desired property holds of the abstraction is as simple as possible. To navigate these conflicting requirements, we first try to abstract the lookup tables with linear abstractions, and see if these simple abstractions are sufficient to prove the specification or to guide the search to a counterexample. If the simple, linear abstractions are insufficient, then we iteratively increase the complexity to a quadratic template, then to cubic, etc. The implementation [6] uses a library of abstraction templates that are indexed by complexity, and iterates through them on each subsequent abstraction attempt.

In the following, we will describe our procedure for computing abstractions from approximations, and how these same abstractions can guide the search for a counterexample when the specification cannot be proved in the first attempt.

3.3.1 Computing Abstractions by Approximation

We use a regression-based procedure to automatically compute a functional interval for each lookup table in the model. First, we fix a parametric template for a function that approximates the lookup table data, and then we will proceed to learn parameter values that allow the function to approximate the lookup table data. Next, we use bisection to search for the smallest offset that can be added and subtracted from the approximation to yield upper and lower bounds for the lookup table function.

We begin by computing an approximation of the lookup table data. Formally, let $f(a, x)$ be a function parametrized by $a \in \mathbb{R}^p$, with the same domain and range as the lookup table function L. We solve a regression problem to find the value of the parameter vector a that

minimizes the mean-squared error over the k breakpoints of the lookup table.

$$\underset{a}{\text{minimize}} \sum_{i=1}^{k} (y^{(k)} - f(a, x^{(k)}))^2$$

Let a^* be the value of a found by this optimization problem. Next, we use the approximation $f(a^*, x)$ to find a functional interval. We begin by setting the offset to some initial value, e.g., $\epsilon = 1$. Then, we use an SMT solver to check whether the lower and upper offset functions $f(a^*, x) - \epsilon$ and $f(a^*, x) + \epsilon$ are lower and upper bounds for the lookup table function over all values in the range of interest $S \subseteq \mathbb{R}^n$. This is equivalent to checking the validity of the following logical formula with an SMT solver.

$$\forall x \in R \,.\, f(a^*, x) - \epsilon \leq L(x) \;\wedge\; L(x) \leq f(a^*, x) + \epsilon$$

Note that the expression for $L(x)$ contains the values of the breakpoints as well as the multilinear interpolation expressions in between the breakpoints of L.

If the validity check fails, i.e., the SMT solver is able to find an $x \in S$ such that the lookup table produces a value outside of the upper and lower bounds, we try again with a larger value of ϵ. If it succeeds, with this value as the upper cap (valid ϵ) and 0 (invalid ϵ) as the lower cap, we then do a bisection search to find the smallest value of ϵ (within some tolerance) such that the offset functions abstract the lookup table. This yields a functional interval

$$A(x) = [f(a^*, x) - \epsilon, f(a^*, x) + \epsilon]$$

such that for all $x \in S$, $L(x) \in A(x)$. This relationship is illustrated in Figure 3.1

3.3.2 Falsification

If the verification attempt does not succeed, it means that a value $x = \hat{x}$ was found such that the abstractions were satisfied, but the specification was falsified. This *candidate counterexample* is not necessarily a true counterexample, since a point that satisfies the abstractions may not satisfy the lookup tables. However, this candidate counterexample

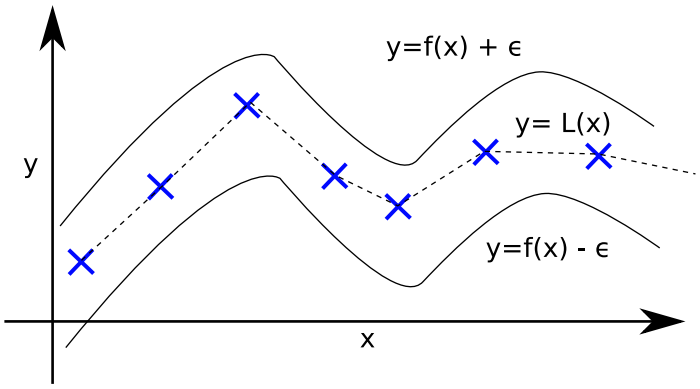


Figure 3.1: Lookup table function $L(x)$ abstracted by upper and lower bounding functions, obtained by shifting an approximation $f(a^*, x)$.

serves as a flag of a region that may contain a true counterexample. It is sensible to search between the breakpoints that contain this counterexample, but note that this point may fall between different breakpoints in different lookup tables, which could potentially lead us to choose intervals from different lookup tables that are inconsistent with each other. To prevent this, instead of simply selecting the two breakpoints that contain the candidate counterexample, we select a small number r of the nearest breakpoints. See Figure 3.2 for an illustration of this mechanism. In our experiments, $r = 3$ or $r = 4$ is usually large enough to prevent inconsistent intervals.

Informally, we construct new lookup tables with only r entries each, and attempt to verify the same model with the reduced lookup tables, this time directly, without abstractions. If the verification succeeds, we know the candidate counterexample was spurious, and can repeat the procedure with a different candidate counterexample. If the verification fails, it provides a true counterexample which can be returned to the engineer as a design flaw that must be fixed. Formally, let $x_j, \ldots, x_{j+n}$ be the n inputs of lookup table L_i. Then, consider the values of these variables in the candidate counterexample $\hat{x}_j, \ldots, \hat{x}_{j+n}$. We wish to extract the r nearest entries along each dimension—suppose they are $x_j^{(k)}, \ldots, x_j^{(k+r)}$ through $x_{j+n}^{(k)}, \ldots, x_{j+n}^{(k+r)}$. Then, construct a new lookup table $\hat{L}_i$ that contains only these breakpoints, and maps them to the same outputs as L_i. Finally, check satisfiability of the

following logical formula.

$$\left(\bigwedge_i \hat{L}_i\right) \wedge \Sigma(x) \wedge \neg S(x) \tag{3.2}$$

If a satisfying instance is found, then that instance is a true counterexample of the original model. If no satisfying instance is found, then we try the procedure with the different candidate counterexamples that are at a distance less than δ from $\hat{x}$. If none of these candidates are true counterexamples, we move on to the next step, which is to refine the abstractions and attempt verification again.

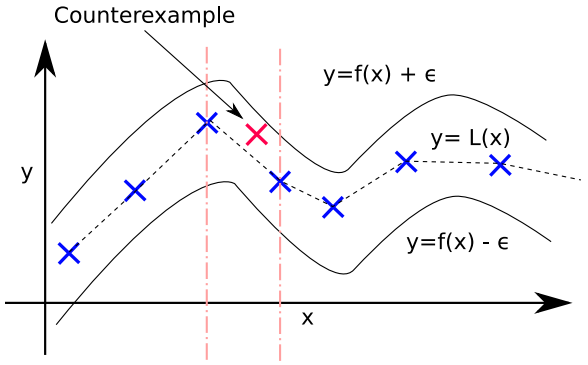


Figure 3.2: An illustration of the falsification process. The red × represents a candidate counterexample. To search for a true counterexample, we construct a reduced table consisting of the two nearest breakpoints, which span the interval between the dashed vertical lines.

3.3.3 Abstraction Refinement

When the SMT solver finds candidate counterexamples, meaning it is unable to prove correctness, and the falsification procedure fails to find a true counterexample, we refine the abstractions and repeat the verification attempt. There are two basic mechanisms by which we refine abstractions: (1) increasing arithmetic complexity of the templates, and (2) increasing the number of cases in the piecewise templates. Increasing arithmetic complexity means moving from linear templates to quadratic templates, higher-order polynomials, or possibly transcendental functions if one is using an SMT solver that supports such functions, such as [55]. Increasing the number of cases in a piecewise template means moving from

a simple equational template to a template with two cases, or from two to three, etc. Our implementation tries both of these techniques at the same time, and keeps the technique that yields the approximation with lowest error. We use the SMT solver z3 in the bisection search procedure to find the minimal offset ϵ that produces a true overapproximation of the lookup table function.

3.3.4 Implementation Details

The above approach is implemented in the tool `Osiris` [6]. We provide relevant details for the implementation below.

3.3.4.1 Proving Specifications

Once each abstraction A_i has been generated for each lookup table L_i ($i = 1, \ldots, k$), we form the following logical formula.

$$A_1(x) \wedge \cdots \wedge A_k(x) \wedge \Sigma(x) \wedge \neg S(x).$$

Then, we invoke the SMT solver z3 to check for satisfiability. If the formula is not satisfiable, z3 has proven that there is no value that satisfies the abstractions and the model constraints but falsifies the specification. Since the abstractions overapproximate the lookup table functions, it follows that the system with the lookup table functions satisfies its specifications. If the formula was satisfiable, we proceed to the falsification stage.

3.3.4.2 Falsification

If a violation of the safety property $\hat{x}$ is found, this does not necessarily mean that the original system violates its specifications. For each lookup table L_i, we the find nearest breakpoints in each lookup table. Then, we try to prove the correctness of the model *only between those breakpoints*. If the verification fails, the result is now a true counterexample, which can be reported to the designer. If no true counterexample is found, we try to compute new abstractions with the next set of templates in the template library.

3.4 Lookup Table Abstraction: Case Study

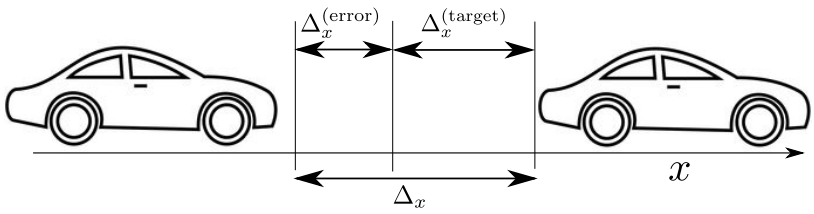


Figure 3.3: Diagram of adaptive cruise control scenario

For our case study, we consider a verification benchmark published by Toyota InfoTechnology Center [124]. This benchmark consists of an adaptive cruise controller along with an online monitor. When enabled, adaptive cruise control regulates the speed of the car so that a target speed is maintained, unless another car is detected at some distance in front, in which case the system tries to maintain a safe distance from the lead car, as shown in Figure 3.3. This controller takes as input the current speed of the car, the distance to the lead car, and the relative speed between the two cars.

The system consists of a cascade of three lookup tables, as shown in Figure 3.4. The inputs to the controller are s, the speed of the controlled car, Δ_x, the distance to the leading car, and Δ_v the relative speed of the two cars.

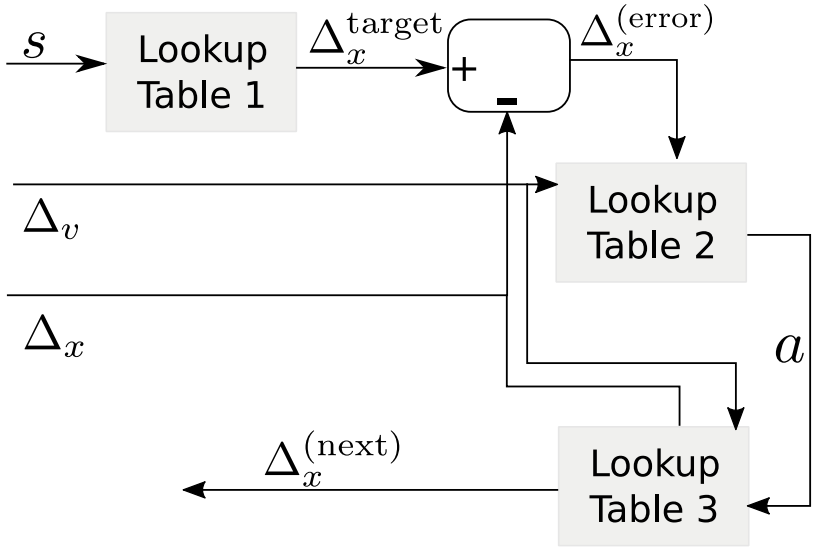


Figure 3.4: Signal-flow model of an ACC controller

The first lookup table uses the current speed s of the controlled car to determine a target

set distance ($\Delta_x^{(target)}$) from the leading car. If the controlled car is moving fast, its braking distance will be larger, which requires the controller to choose a larger following distance. $\Delta_x^{(error)}$ is the difference between the target following distance and the chosen following distance, and the second lookup table uses $\Delta_x^{(error)}$ together with the relative velocity Δ_v to choose an acceleration a. The third lookup table behaves as an online monitor. In practice, a monitor lookup table would be produced by recording observations of a physical component. For this example, the monitor was generated by computing the future distance between the two cars after 0.1 seconds, given the current distance, relative velocity, and chosen acceleration. This monitor assumes that the lead car will not change its velocity within the next 0.1 seconds.

The property we wish to prove is that the online monitor will never predict a future distance that is negative, i.e., it will never predict that the cars will crash. This does not mean that the closed-loop system with the real automotive dynamics will not crash, since that would require analyzing the continuous-time differential equations. However, industrial controllers are frequently equipped with online monitors that predict or prevent dangerous conditions, and checking that the controller satisfies its monitor is valuable, as it prevents any abnormal behavior as long as system integrity is preserved. However, our approach is not limited to analyzing properties based on a monitor, and can analyze general properties expressed in first-order logic over the variables of the model.

The first lookup table is one-dimensional and contains 21 breakpoints, resulting in 22 possible interval values. The second lookup table is two-dimensional, and has a total of $1,232$ possible interval values. The third lookup table is three-dimensional and has $2,856$ possible values. A brute-force attempt at proving correctness would need to consider all possible combinations of lookup table values. Considering all possible combinations of internal values leads to a total of $77,409,024$ proof cases.

The model is translated to first-order logic to use an SMT solver to check the validity of the specification. The logical formulas that represent the lookup table can be directly constructed by multilinear interpolation on the public benchmark files. The translated first-order logic constraints are:

$$0 \leq \Delta_x \leq 180, \qquad -50 \leq \Delta_v \leq 50, \qquad 0 \leq s \leq 180,$$
$$\Delta_x^{(target)} = \texttt{LookupTable}_1(s), \qquad \Delta_x^{(error)} = \Delta_x - \Delta_x^{(target)},$$
$$a = \texttt{LookupTable}_2(\Delta_x^{(error)}, \Delta_v), \qquad \Delta_x^{(next)} = \texttt{LookupTable}_3(\Delta_x, \Delta_v, a).$$

The constraints on Δ_x, Δ_v, and s are assumptions on the bounds of these inputs, and the system cannot be enabled if these bounds are not met. Similarly, commercial adaptive cruise control systems cannot be used if the speed of the controlled car is too slow. When attempting to directly verify the model by translating it into first-order logic constraints and using z3 to check for a violation of the specification, z3 does not terminate after 48 hours. When we ran this model, a counterexample was found in 1 minute and 50 seconds, as follows:

$$s \mapsto 31.0, \qquad a \mapsto -2.0, \qquad \Delta_v \mapsto -4.0, \qquad \Delta_x \mapsto 0.03125,$$
$$\Delta_x^{(error)} \mapsto -30.97, \qquad \Delta_x^{(target)} \mapsto 31.0, \qquad \Delta_x^{(next)} \mapsto -0.00865.$$

The meaning of this counterexample is that the cars start at a distance Δ_x of about 3 cm, with a relative velocity of -4 m/s, i.e., the controlled car is moving 4m/s faster than the lead car. The controller brakes by applying a negative acceleration of $a = -2$ m/s^2, but the situation is already too dangerous and the cars have a minor crash, with the controlled car being 0.8 cm further than it should be.

To measure the run-time of our verification technique, we relaxed the specification to $\Delta_x^{(next)} \geq -2$. With this relaxed property, the monitor no longer tries to completely prevent collisions, but simply to reduce their severity. Of course, this is not a controller that could be deployed for a commercial automotive system, it is simply for benchmarking of our tool. This relaxed property was provable in 30 seconds, which compares favorably with an analysis time of approximately four hours in [69] on a machine with the same specifications. The case study computations were carried out on a machine with 44 cores, with available hyperthreading to 88 threads and 256 GB of RAM.

Computed abstractions The abstraction computed for lookup table 1 consists of a linear function, shifted above and below the lookup table data:

$$A_1 = [1.31s - 4.0315 - \epsilon_1, 1.31s - 4.0315 + \epsilon_1],$$

We have deliberately left the constants un-simplified. The constant $\epsilon_1 = 34.1797$ is useful because it represents the largest error between the abstraction and the lookup table itself. Thus, we can compare which lookup tables are being abstracted with more or less fidelity by looking at the value of ϵ. The abstraction computed for lookup table 2 is a linear function, and has the form

$$A_2 = [f_2 - \epsilon_2, f_2 + \epsilon_2]$$

where $f_2 = 0.023843553\Delta_x^{(error)} + 0.091889\Delta_v - 0.51779$, with $\epsilon_2 = 3.90625$. The abstraction computed for lookup table 3 is a linear function.

$$A_3 = [f_3 - \epsilon_3, f_3 + \epsilon_3]$$

where $f_3 = 0.99876517\Delta_x + 0.00795821\Delta_v - 0.0016369$, with $\epsilon_3 = 0.5859375$.

3.5 Constraint Solving: Overview

Next, we develop a similar technique for constraint problems corresponding to large sets of nonlinear constraints. For a set of constraints $A = \{A_1, \ldots, A_m\}$, the constraint satisfaction problem is to find a point satisfying the constraints, or to prove that the problem is unsatisfiable. Our approach is to decompose the original set of constraints $A = \{A_1, \ldots, A_m\}$ into subsets $\mathbf{A}_1, \ldots, \mathbf{A}_k$, and then learn antecedents and consequents for each subset. We then use these antecedents and consequents to solve the original constraint satisfaction problem. The details of our learning procedure will be described later in this section, but for now we simply note that it relies on obtaining satisfying and falsifying instances and learning classifiers from them that are antecedents or consequents.

Let α_j be the learned antecedent for the subset $\mathbf{A}_j$, and let γ_j be its consequent. Then,

we solve the sets of constraints $\alpha = \{\alpha_1, \ldots, \alpha_j\}$ and $\gamma = \{\gamma_1, \ldots, \gamma_j\}$. If the antecedents α are satisfiable by some instance $\hat{x}$, then $\hat{x}$ satisfies the original constraint set, by definition of antecedence. Conversely, if the consequents γ are unsatisfiable, then the original set A is also unsatisfiable. For if A were satisfiable, then each $\mathbf{A}_j$ would be satisfiable, and so would each γ_j, by the definition of consequence. Alternatively, if α is unsatisfiable and γ is satisfiable, then we cannot conclude anything about A. We must refine our antecedents and consequents and try again.

Our technique provides performance improvements under the following premises:

1. The subsets of constraints $\mathbf{A}_j$ are simple enough that they can be solved quickly and repeatedly to obtain satisfying and falsifying instances.

2. The learned antecedents and consequents are simple enough that checking the properties of antecedence and consequence is fast.

3. The learned antecedents and consequents are accurate enough to establish the existence or non-existence of a solution.

The above requirements are inherently conflicting. Greater simplification is obtained by considering larger subsets $\mathbf{A}_j$, but these larger subsets are harder to solve quickly to obtain samples. Similarly, simpler antecedents and consequents will be easier to check, but will provide poorer approximations to solve the original problem.

To illustrate the proposed approach, consider the problem of solving the constraint sets A and B shown in Figure 3.5. The region arising from constraint set A is underapproximated by the region A'. The constraint set A' is now solved in conjunction with B to find a solution satisfying A and B. The learned constraint set A' has a simpler form, enabling faster computation.

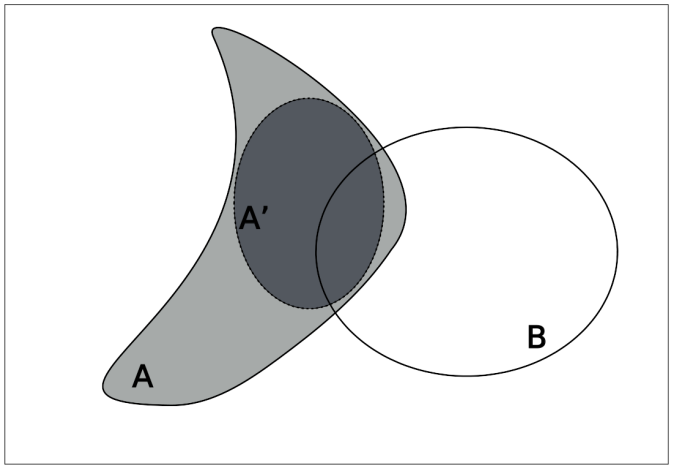


Figure 3.5: A point satisfying the constraint sets A and B can be computed by replacing A with a simpler *underapproximation* A', and then finding a point satisfying A' and B.

3.6 Constraint Solving: Learning Abstractions

3.6.1 Semi-Soft SVM

As a supervised learning approach, SVMs have been known to perform effectively in learning classifiers. Two types of SVM are commonly employed: hard margin and soft margin. Hard-margin SVM approaches do not allow for any misclassification, but may be infeasible if the data points are not linearly separable. On the other hand, soft-margin SVM approaches allow for misclassification of some points, but they do not serve our purpose. Since in general an antecedent will need to exclude all falsifying instances and a consequent will need to include satisfying instances, soft-margin SVMs are not suitable for our purposes.

Instead, we use a semi-soft SVM, where we allow only samples of one type to be misclassified. When searching for an antecedent (underapproximation of satisfying instances), we allow some positive examples to be misclassified. Similarly when searching for a consequent, we allow some negative examples to be misclassified, resulting in an overapproximation of the satisfying instances. Asymmetric SVMs have been of interest to minimize cases of false negatives (or false positives) in certain applications. Some efforts directed at asymmetric learning [122, 123] aim to minimize the false negatives but for

learning consequents we seek to eliminate false negatives.

Use of a semi-soft SVM yields a quadratic program. To abstract the constraints $\mathbf{A}_j$, we label its satisfying instances as $+1$ and its falsifying instances as -1. To compute a consequent, we set up the optimization problem to necessarily yield positive labels for the positive instances, while simply making a best effort attempt to provide negative labels for the negative instances.

Let X^+ be the set of satisfying instances of the subset of constraints $\mathbf{A}_j(x)$ and X^- the set of falsifying instances of $\mathbf{A}_j(x)$. X^+ and X^- are obtained by the sampling procedure described in Section 3.6.2. We will search for a consequent by the following optimization problem, which learns the classifier $g(x) = sgn(w_0 + w^T x)$:

$$\begin{aligned} \min_{w,w_0,e_b} \frac{1}{2}\left(w_0^2 + w^T w + \lambda \sum_{n=1}^{N^-} e_n\right) \\ s.t \qquad y^-\left(w_o + w^T x_b^-\right) \geq 1 - e_b, \\ y^+\left(w_o + w^T x^+\right) \geq 1, \\ e_b \geq 0, \\ \forall x^+ \in X^+, \\ \{x_1^-, x_2^-, \ldots, x_{N^-}^-\} = X^-, \\ b = 1, 2, \ldots, N^-, \end{aligned} \tag{3.3}$$

where $N^- = |X^-|$ and $N = |X^+ \cup X^-|$. Here e_b are the slack variables allowing for misclassification of the points in X^-. Notice that this deviates from the standard soft-margin SVM in the sense that only one type of data points are assigned slack variables. The labels $y^+ = +1$ and $y^- = -1$ correspond to points in X^+ and X^- respectively. Note that this optimization problem provides hard constraints for classifying positive points, but soft constraints for classifying negative points.

Proposition 8. For any given sets of points X^+ and X^-, the semi-soft SVM as formulated in (3.3) finds a hyperplane $w_0 + w^T x = 0$ such that $\forall x^+ \in X^+$, $w_0 + w^T x^+ > 0$.

Proof. For $x^+ \in X^+$, from $y^+ (w_o + w^+ x^+) \geq 1$ we have $w_0 + w^T x^+ \geq 1 > 0$.

The minimization problem in equation (3.3) is a quadratic program (QP). A solution to the constraints of the QP is $w_0 = 1, w = 0$ and $e_b = 2$, implying that the QP has a non-empty feasible set. Therefore, equation (3.3) always returns a hyperplane that has the above property of correctly classifying the points in X^+. □

The case for computing an antecedent can be derived in a straightforward manner, by requiring that the classifier correctly provide negative labels for the negative instances, while simply making a best effort attempt to provide positive labels for the positive instances.

3.6.2 Sampling for Learning

A key aspect of this problem is sampling points that reasonably cover both the regions satisfying the constraints and the regions that do not satisfy the constraints to learn meaningful classifiers. One such simple sampling strategy follows.

We will use a distance metric $d(\cdot, \cdot)$ to enforce spacing between samples. For the experiments in this chapter, the distance function is the ℓ^2-norm. We begin by choosing a large radius R_0, and sample a set of points $C = \{c_1, \ldots, c_k\}$ that are separated by a distance of at least R_0. These samples will serve as centers for circles that we will sample from at the next iteration.

Now, consider a smaller radius $R_1 (< R_0)$. Define circles with radius R_0 around each of the sampled points $\Omega_i = \{x | d(x, c_i) \leq R_0\}$. In each Ω_i, sample points that are at least R_1 apart. In this manner we can iterate through a sequence of decreasing radii $R_2, R_3 \ldots R_l$ by defining circles around the sampled points with radius R_i and sampling points from these circles that are at least R_{i+1} apart and so on.

The radii R_k are chosen so that the points sampled in a particular layer are distributed around the feasible region (the intersection of the feasible region for the problem and the circles defined by us) for that layer. Given a fixed number of samples m to be sampled in a layer, the optimal radius for that layer can be determined by a bisection search to find approximately the largest radius that gives m samples per layer. This implies that there exists no other point in the feasible region that can be sampled that is at a distance greater

than r_k from the other sampled m points. Further increasing the radius would imply that we cannot find m such points, and if we can find more than m points we increase the radius until only m are found.

This approach provides good coverage in our experiments. However, even if the initial samples are poorly distributed, the CEGAR loop described in Section 3.6.3 iteratively searches for antecedents and consequents, and only terminates once provable abstractions are found. As a result, the quality of the initial samples only has an effect on performance, but does not compromise soundness. The approach is more formally described in Algorithm 2. The term 'Sample' in Algorithm 2 refers to a query to the constraint solver to find a feasible point satisfying the constraints. Though Algorithm 2 provides no formal guarantees about m samples being found during each iteration, the learning can be performed with fewer than m samples. When the constraints have no solution and no sample can be found, the classifier learned is `False`.

Besides good coverage, an advantage of this approach is that it prevents the explosive growth of the formula used to sample using an SMT solver. Once the first set of samples is generated, the procedure is parallelized to generate samples in the balls around each of these points since the sampling in the individual balls is independent of that in the others.

3.6.3 Counterexample-Guided Abstraction Refinement(CEGAR)

Counterexample-guided abstraction refinement (CEGAR) was first proposed in [28] for refining abstractions of control structures in programs. We develop our algorithm in the paradigm of CEGAR.

Once the classifier $g(x)$ has been learned by the semi-soft SVM, it still remains to be validated that the classifier learned is an overapproximation (underapproximation) of the constraint set we seek to abstract. Let A_i be the conjunction of the constraints in $\mathbf{A}_i$. To verify that the classifier learned is indeed a consequent (antecedent) of the constraint set $\mathbf{A}_i(x)$, we need to check the consequence condition $A_i(x) \implies g(x) > 0$ (for showing antecedence $g(x) > 0 \implies A_i(x)$). If the negation of this formula is not satisfiable, i.e, there is no assignment to x over which the formula interprets to `False`, we have learned a

Algorithm 2: Sampling Procedure

Input : Set of constraints $A(x)$, $k, l, m, R_1, R_2, \ldots, R_n$
Output Sets of points X^+ satisfying $A(x)$
:

$X^+ = \emptyset$, $\bar{X}^+ = \emptyset$;
for *j=1:m* **do**
    Sample $p \vdash A(x)$ s.t. $\forall s \in \bar{X}^+ . d(p, s) > r_{i+1}$;
    $\bar{X}^+ = \bar{X}^+ \cup p$;
end
for *i=1:l-1* **do**
    $\underline{X}^+ = \emptyset$;
    for $q \in \bar{X}^+$ **do**
        $X^+ = \emptyset$;
        for *j=1:m* **do**
            Sample $p \vdash A(x)$ s.t. $d(p, q) < r_i$ and $d(p, s) > r_{i+1} . \forall s \in X^+$;
            $X^+ = X^+ \cup p$;
        end
        $\underline{X}^+ = \underline{X}^+ \cup X^+$;
    end
    $\bar{X}^+ = \underline{X}^+$;
end
$X^+ = \underline{X}^+$;
return X^+ ;

Algorithm 3: Algorithm to learn abstractions for sets of nonlinear arithmetic constraints

Input : Set of constraints $A(x)$, parameters m, k
Output $g(x)$ such that $A(x) \implies g(x) > 0$
:

Sample set of points X^+ with $A(x)$ as input to Algorithm 2 ;
Sample set of points X^- with $\neg A(x)$ as input to Algorithm 2 ;
Initialize $g_0(x)$ = **TRUE** , iter = 0;
Set $X = X^+ \cup X^-$;
Map the X to a higher-dimension feature space ;
Cluster points in X^- into $\mathcal{N}$ clusters ;
Learn a classifier separating each cluster in X^- from X^+ using the semi-soft SVM;
Set $g(x) = g_0(x) \wedge$ conjunction of the $\mathcal{N}$ classifiers ;
if $A(x) \implies g(x) > 0 \ \wedge$ iter $\leq k$ *;*
then
    return $g(x)$;
else if iter $\leq k$ **then**
    iter=iter+1;
else
    $g_0 = g(x)$;
    iter=0;
Sample a set of points X^* with $\neg(A(x) \implies g(x) > 0)$ as input to Algorithm 2;
$X^+ = X^+ \cup X^*$;
go to 4 ;

consequent $\gamma_i(x) \equiv (g(x) > 0)$ (respectively, antecedent $\alpha_i(x) \equiv (g(x) > 0)$). Otherwise, we sample points that violate the formula as proposed in Algorithm 2, add them to the training set, and learn a new classifier. This procedure is iteratively repeated until we have learned a consequent (antecedent). Note that although Algorithm 3 may fail to terminate, it will terminate only if it has found a consequent for $\mathbf{A}_i(x)$.

3.6.4 Boosting

In some problems, the CEGAR procedure might fail to converge after a large number of iterations or learn poor approximations of the original problem. This can occur if the constraints to be approximated represent regions in space that cannot be approximated by a single classifier. This section describes how to sequentially learn classifiers that in conjunction form a better relaxation.

As a first step, the falsifying points are partitioned into clusters using a clustering algorithm (K-means for all examples presented in this chapter). A classifier is learned from each cluster separating it from all the feasible points. These classifiers in conjunction form a first approximation of the constraints. Next, a few iterations of CEGAR are run where this approximation is refined. Subsequently, collect the misclassified falsifying points and learn a new classifier using the misclassified points and the satisfying points in a similar manner. Note that none of the feasible points are misclassified by the construction of the semi-soft SVM.

During the CEGAR step that learns the new classifier, the condition $A_i(x) \implies \gamma_i(x)$ is checked where now $\gamma_i(x)$ is the conjunction of the most recent classifier learned and the previous classifiers. The classifier learned at the end of these CEGAR iterations is the conjunction of the two classifiers (which in turn are a conjunction of classifiers).

This differs from conventional boosting [111], where the classifier learned is a weighted sum of an ensemble of classifiers, and cascade SVMs [60], where the classifiers are learned in parallel and then combined into a single one.

Example 5. To demonstrate the approach described above consider the following constraint

$$\begin{aligned} A = & x^2 + y^3 \leq 4 \wedge x \geq -4 \\ & \wedge\ x^2 + y^3 + 30y \leq -20 \\ & \wedge\ x^5 + 3xy - 2x \leq 5. \end{aligned} \tag{3.4}$$

Suppose we want to find a point that lies outside A but satisfies a constraint C, i.e, we are trying to solve $\neg A \wedge C$.

We first seek to learn an underapproximation of the feasible region ($\neg A$) or an overapproximation of the infeasible region (A) with quadratic classifiers, since a collection of quadratic constraints is easier to solve than a collection of higher-order polynomial constraints. Recall that by negating the overapproximation of A, we can obtain an underapproximation of $\neg A$.

For constraint A we obtain satisfying and falsifying instances using Algorithm 2. Next, the sampled instances are transformed to a higher dimensional space by a feature map [17], which lets us learn quadratic classifiers in the higher dimensional space by the semi-soft SVM procedure described in Section 3.6.1. An antecedent for $\neg A$ is learned as described in Algorithm 3. Recall that the antecedence is verified using an SMT solver in Algorithm 3.

Here, the number of clusters is set at 2 and we repeat CEGAR until it converges. We learn an abstraction that is a conjunction of two quadratic classifiers. The abstraction is $A' = 0.109xy - 0.788x + 0.915x^2 + 0.935y + 0.135y^2 \geq 1.03 \wedge 5.077 - 0.338xy + 6.001x + 1.36x^2 + 0.887y + 0.257y^2 \geq 0$. A' underapproximates $\neg A$. To solve $\neg A \wedge C$, we solve $A' \wedge C$. Figure 3.6 shows the original set of constraints (red). The infeasible region (A) is shaded. The classifiers learned are shown in the graph (blue). The feasible area is underapproximated by the intersection of the regions outside each of the classifiers. The solution space for the problem is restricted to $x \in [-2\pi, 2\pi]$ and $y \in [-2\pi, 2\pi]$.

3.6.5 Decomposition Methods

This section describes the heuristics that we use to select which constraints should be considered together to learn an abstraction. The first level of decomposition is based on the

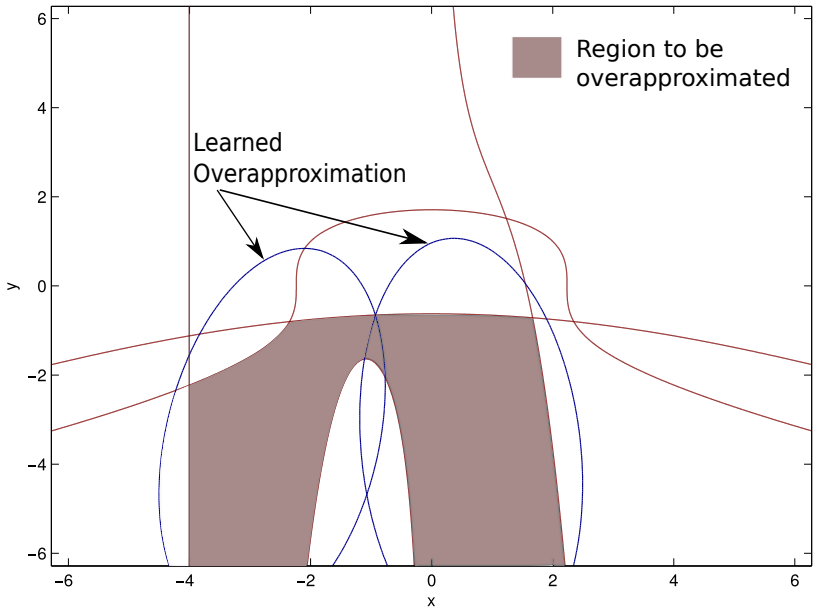


Figure 3.6: Quadratic Overapproximation of the infeasible region.

Hamming distance.

Hamming Decomposition: This heuristic groups together constraints that have many common variables—i.e., those constraints whose sets of variables, treated as vectors, have a small Hamming distance. Let $FV(\cdot)$ be the function that takes a clause and returns the set of free variables of that clause. Then, the Hamming distance between constraints c_m and c_n can be computed as

$$H(c_m, c_n) = |(FV(c_m) \setminus FV(c_n)) \cup (FV(c_n) \setminus FV(c_m))|.$$

We assume that a maximum distance bound θ is given, such that any two constraints with a Hamming distance less than or equal to θ will be grouped together for abstraction. Our implementation loops over the constraints; starting with the first constraints c_0, it chooses all constraints c_k such that $H(c_0, c_k) \leq \theta$. After the first pass, it chooses the next constraint that was not grouped and loops over the remaining constraints.

If we group the constraints into a few classes with many constraints each, then sampling each class to generate abstractions will be computationally difficult. In the limit case, if all constraints are grouped into a single class, then choosing a single satisfying instance corresponds to solving the original satisfiability problem.

Bounded-size decomposition: In some cases, it may be the case that the earlier decomposition did not produce classes that are sufficiently small to ensure that sampling can be performed quickly. In this case, we set a bound n, and divide the large classes into smaller subsets with $\leq n$ constraints each.

3.7 Constraint Solving: Experiments

This section details the results of experiments on various benchmark sets on a 32-core AMD Opteron machine at 2.3 GHz, with 96 GB of RAM.[1] A timeout of 200 seconds is set for abstracting each subset, if the abstraction procedure fails to finish in this period the constraints are used as is.

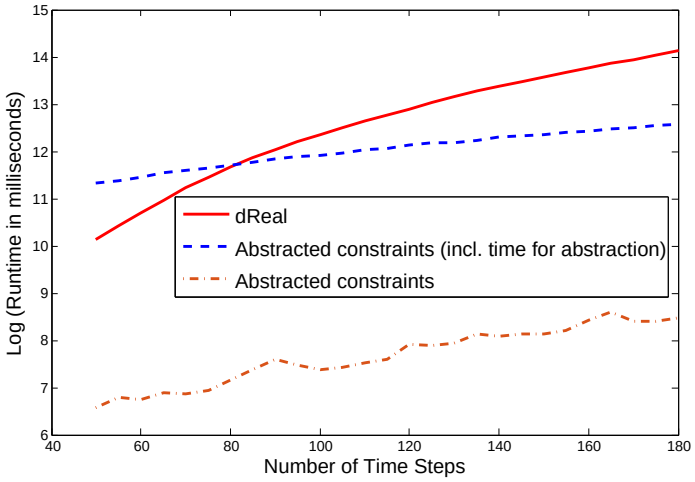


Figure 3.7: Performance – Car Reachability

Model predictive control for a mobile robot First, we consider a benchmark motivated by practical problems in mobile robotics. The model predictive control problem is to find a sequence of control inputs so that a mobile robot can reach a specific position. We use a

[1]Benchmark problem instances can be found at `https://github.com/dathath/IJCAI_2017_SD`

Dubins car dynamics model with additional nonlinear terms:

$$\begin{aligned} a_{\mathrm{t}} + c_1(a_t + x_t/y_t) &\geq a_{t+1}, \\ y_t + c_2\left(\cos(a_t) - y_t^2\right) &\geq y_{t+1}, \\ x_t + c_3\left(\sin(a_t) - x_t^2\right) &\geq x_{t+1}. \end{aligned} \tag{3.5}$$

The constraints encode the problem of starting from an initial point and the objective is to determine if a target polytope can be reached, i.e., $(x_{final}, y_{final}) \in ([a, b] \times [c, d])$.

The problem's scale can be varied by changing the number of time-steps. All the problem instances are satisfiable. Figure 3.7 shows a plot of the run times for dReal, our abstraction approach, and the time required to check abstractions alone. We cannot compare with un-abstracted performance of z3, because z3 does not support solving over sines and cosines. For this same reason, the sampling and abstraction checking are carried out with dReal, though we solve the simplified problem with z3, because in this example the resulting abstractions are conjunctions of linear constraints, and z3 typically performs better than dReal at linear constraints. The classifier learned for this example is a conjunction of several linear classifiers, one linear classifier learned during each step of CEGAR. The Hamming distance threshold here was set to 4, which results in subsets with 3 constraints each. Since sampling these constraints is fast, further decomposition is not required. The abstracted constraints are always solved faster than solving the constraints directly, which could be useful in an application scenario in which abstractions can be cached and re-used. For large problems, the abstraction-based approach (including the time for abstraction) performs considerably better than solving directly with dReal.

Energy-efficient FPGA design The constraint sets here correspond to the search for valid probabilistic encoder expressions in a value-deviation-bounded serial encoding technique [107]. The constraint satisfaction problem is to find assignments of probabilities for a set of Bernoulli random variables such that they satisfy the constraints for valid encoder family formulations. The constraint sets here correspond to sets of higher-order polynomials. The Hamming distance threshold is set to 3 and n is set at 5. The number of clusters was set to

Solver	No. Of Benchmarks Solved	Average Time for Benchmarks Fully Abstracted (ms)	Avg. Number of Constraints Input Per Instance	Avg. Degree of Constraints Input to Solver
z3	29	1793.4	~32	~5.93
dReal	28	1788.8	~32	~ 5.93
Post-Abstraction (with dReal)	20	133.3	~11.5	~ 2.06

Table 3.2: Comparison on benchmarks for which the abstraction procedure completes

2. The classifiers learned were of the form:

$$g(x) = \left(\sum_i \left(a_i x_i^2 + b_i x_i\right) + c_0 > 0\right) \wedge \left(\sum_i \left(d_i x_i^2 + e_i x_i\right) + f_0 > 0\right).$$

Sphere packing in high-dimensions In this set of benchmarks, we solve for a point $x \in \mathbb{R}^{15}$ in the intersection of the exterior of a set of randomized high-dimensional spheres of the form:

$$\sum_{j \in \mathrm{I}} (x_j - a_j)^2 \leq 30, \tag{3.6}$$

where $I \subseteq \{1, 2, \ldots, 15\}$ is a randomly chosen set that determines the constrained variables $\{x_j : j \in I\}$ and cubes bounding the search region ($|x_k| \leq 30$). For each sphere, the parameters $a_j : \forall j \in I$ are randomly chosen in the range $[-50, 50]$. Figure 3.8 shows a plot of the different runtimes for varying number of constraints. The time for solving the abstracted constraints is significantly smaller than the time for solving directly with dReal and z3 and the time for abstraction scales almost linearly since the time for abstracting each subset is independent of the total number of constraints. The parameters for the learning procedure are the same as those from the FPGA design problem.

Hong family. These sets of benchmarks correspond to the Hong family of benchmarks [73]. We restrict the solution space to the positive quadrant. The problem instances are

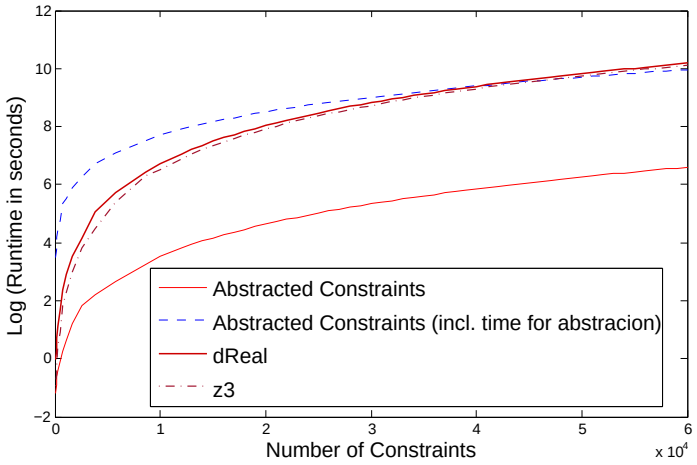


Figure 3.8: Performance – Randomized spheres

always `unsat` by construction. A parameterized generalization of the problem is:

$$\prod_{i=1}^{n} x_i > 1, \qquad \sum_{i=1}^{n} x_i^2 < 1. \tag{3.7}$$

To facilitate learning abstractions, we rewrite the problem by replacing the quadratic constraint with constraints of the form $x_i^2 + x_{i\oplus 1}^2 \leq z_j$ and $\sum_j z_j \leq 2$. This makes the sampling process easier because each of these constraints have to be sampled only in three dimensions. The Hamming threshold is set to 3 and bounded-size decomposition is not used here. The classifiers learned are a conjunction of linear classifiers as in the car benchmark. Figure 3.9 shows the runtimes. z3 does not scale beyond low dimensions, while dReal almost scales linearly. The abstraction-based scheme does reasonably well at low dimensions but at higher dimensions the relaxations computed slow down dReal, interfering with the δ-complete satisfiability procedures used in dReal.

3.8 Complexity and Discussion

Since the benchmarks contain trigonometric and highly nonlinear functions, even the relaxed versions of the problem that admit solutions are NP complete [54]. Existing solvers run into scalability issues when dealing with large constraint-solving instances as in [58].

In our approach, the number of samples grows exponentially with the number of variables

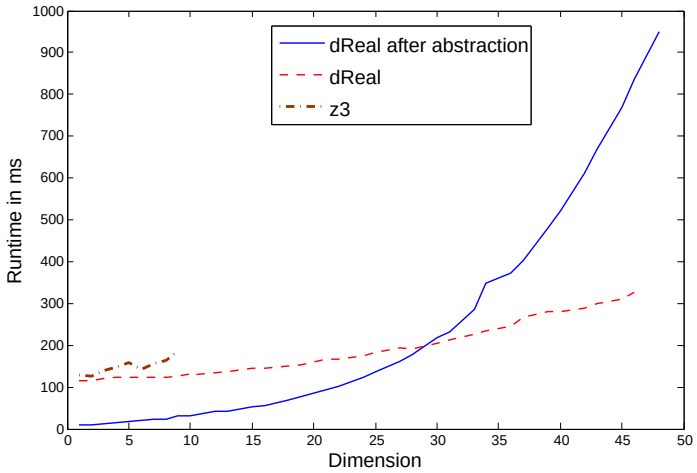


Figure 3.9: Performance – Hong Family

in the constraints. But by restricting to small subsets of the original set of constraints, this exponential growth can be capped. Lloyd's algorithm [85] for K-means clustering is a linear-time algorithm and the semi-soft SVM results in a QP that is solvable in polynomial time. Verifying that the classifier learned from a small subset of constraints is a consequent (or an antecedent) is a reduced problem whose complexity is much smaller than solving the entire large set of constraints, since the complexity of constraint-solving scales exponentially. We use a sequence of polynomial-time algorithms to learn an approximate simpler instance of the constraint satisfaction problem.

From the experiments, we observe that though the abstraction procedure is not complete it can improve the handling of large sets of complex constraints abstracting certain subsets of the constraints. The experiments demonstrate the potential of the learning-based abstraction approach in improving the scalability of constraint solvers.

3.9 Data-driven Verification: Preliminaries

In both Sections 3.2 and 3.5, we considered systems that are complexity characterized by first-order logic formulas. However, in more general settings in the real-world, it is often difficult to obtain accurate models of the environment with the controller interacts. This section (based on [23]) develops an approach on providing meaningful guarantees in such

scenarios. Consider a dynamical system Σ described by differential or difference equations

$$x^+ = f(x, u, d), \tag{3.8}$$

where $x \in \mathcal{X}$ is the state, $u \in \mathcal{U}$ is the control input, $d \in \mathcal{D}$ is environment input, and $\mathcal{X}$, $\mathcal{U}$, $\mathcal{D}$ incorporates the physical limits of the variables. A run of Σ is an indexed family σ consisting of 3-tuples of the form $\sigma_t = (x(t), u(t), d(t))$, satisfying the dynamics equation. If σ is a run of Σ, we will also write $\sigma \models \Sigma$. A run can be infinite or finite with horizon T.

3.9.1 Signal Temporal Logic

To express desired properties for the system, we use the formalism of *signal temporal logic* (STL) [40], an extension of linear temporal logic to vector-valued signals. For any $a, b \in \mathbb{R}$, we will denote by $[a, b]$ the closed interval $\{x \in \mathbb{R} \mid a \leq x \leq b\}$. STL formulas can be built recursively as

$$\varphi \triangleq \texttt{True} \mid p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \mathbf{U}_{[a,b]} \psi,$$

where p is an atomic predicate of the form: $p \triangleq f(\sigma(t)) > 0$ for some $f : \mathbb{R}^m \to \mathbb{R}$. We write $(\sigma, t) \models \varphi$ to indicate that φ holds for $\sigma(t)$. The satisfaction of a signal σ at time t for any of the building block formulas in (3.9.1) is defined in the obvious way, except perhaps the one with the "until" operator $\mathbf{U}$, which is given as

$$(\sigma, t) \models \varphi \mathbf{U}_{[a,b]} \psi \Leftrightarrow \exists \tau \in [t + a, t + b].(\sigma, \tau) \models \psi \wedge \forall \tau' \in [t, \tau].(\sigma, \tau') \models \varphi.$$

For convenience, we can define the "eventually" $\Diamond$ and "always" $\Box$ operators as $\Diamond_{[a,b]}\varphi \triangleq \texttt{True}\mathbf{U}_{[a,b]}\varphi$ and $\Box_{[a,b]}\varphi \triangleq \neg(\Diamond_{[a,b]}\neg\varphi)$ such that $(\sigma, t) \models \Diamond_{[a,b]}\varphi$ if and only if φ is satisfied at least once within a time window of length $b - a$, a time units from t while $(\sigma, t) \models \Box_{[a,b]}\varphi$ requires that φ should always be satisfied within that time window.

3.9.2 Random Convex Program

We analyze the proposed approach using random convex programs (RCP) [20], which we introduce here. Let $P[K]$ denote a (minimization) optimization problem with a known objective function and constraint set K, and let $\text{Obj}[K]$ denote the optimal objective value of $P[K]$. A constraint k is a supporting constraint if $\text{Obj}[K\backslash\{k\}] < \text{Obj}[K]$. The setup for an RCP is the following:

$$\begin{aligned} &\min\ J(\alpha) \\ &\text{s.t. } \alpha \in Q(\delta_i), \forall \delta_1, ..., \delta_N \text{ i.i.d samples of } \delta, \end{aligned} \tag{3.9}$$

where $\delta \in \Delta$ is a random variable in the space Δ and $\{\delta_i\}$ are independently identically distributed samples of δ, $\alpha \in \mathbb{R}^n$, $Q(\delta_i) \subseteq \mathbb{R}^n$ is a convex set determined by δ_i, and $J(\alpha)$ is a convex objective function. Each δ_i would pose a convex constraint on α. If we randomly draw N samples of δ, and denote it as $\omega \doteq \delta_{1:N} \in \Delta^N$, then let $Q(\omega) \doteq \bigcap_{i=1}^{N} Q(\delta_i)$, define

$$V^*(\omega) = \mathbb{P}\left\{\delta \in \Delta : \text{Obj}([Q(\omega), Q(\delta)]) > \text{Obj}[Q(\omega)]\right\}, \tag{3.10}$$

which is the probability that an additional sample added on top of ω would change the objective value of the original optimization with constraints determined by ω. [20] gives upper bound on $\mathbb{P}(V^*(\omega) \geq \epsilon)$ given $1 \geq \epsilon > 0$ for a randomly drawn sequence of samples ω. We recall the following relevant lemma from [20]:

Lemma 9. *Consider the random convex program in* (3.9) *where* $\alpha \in \mathbb{R}^n$. *When* $N \geq \zeta$, $\mathbb{P}\left\{\omega \in \Delta^N : V^*(\omega) > \epsilon\right\} \leq \Phi(\epsilon, \zeta - 1, N) \leq \Phi(\epsilon, n, N)$, *where* ζ *is the Helly dimension denoting the maximum number of supporting constraints, which is bounded by* $n + 1$.

$$\Phi(\epsilon, k, N) = \sum_{j=0}^{k} \binom{N}{j} \epsilon^j (1-\epsilon)^{N-j} \tag{3.11}$$

is the cumulative distribution of a binomial random variable, that is, the probability of getting no more than k *successes in* N *Bernoulli experiments with success probability* ϵ.

This is Theorem 3.3 in [20], which shows that the result of the RCP is likely to be true

for unseen δ drawn from the same distribution under large N and small n. We will revisit this lemma in Section 3.10 to prove probabilistic correctness of the proposed method.

3.10 Data-driven Verification: Approach

Given a dynamical system Σ, a specification about the initial condition φ_0, a controller φ_c and a performance specification φ_p, the goal is to verify that φ_p is satisfied by all the runs of the system, with high probability, when the control input and initial condition satisfy φ_0 and φ_c. However, since the system is interacting with the environment, φ_c and φ_0 alone typically do not imply φ_p, i.e., the verification fails trivially assuming that the environment can choose arbitrary behaviors. Therefore, we look for an assumption of the environment φ_e that explains the observed data with high probability and such that $\Sigma \wedge \varphi_0 \wedge \varphi_e \wedge \varphi_c \Rightarrow \varphi_p$ holds, where the implication is understood as

$$\forall (x(t), u(t), d(t)), ((x(t), u(t), d(t)) \models \Sigma \wedge \varphi_0 \wedge \varphi_c \wedge \varphi_e) \Rightarrow \left((x(t), u(t), d(t)) \models \varphi_p\right).$$

We propose a framework that learns φ_e by using a falsification procedure in the loop, in addition to data of interaction between the system and the environment. Figure 3.10 depicts an overview of the framework being proposed. The falsification module takes a fixed controller, the reactive bound of the environment, and the specification φ_p as inputs, and either returns traces of the system evolution and the environment behavior that falsify φ_p under the given controller, or returns a flag saying that no falsifying trace could be found. In particular, we use the tool S-Taliro as an oracle falsifier, which uses stochastic sampling and can handle STL formulas: see [4] for details. When the falsification returns no trace, the procedure terminates and the verification is successful. When falsifying traces are found, they are fed into the reactive modeling module where the positive traces collected from the actual interaction with the environment and the negative traces from the falsification module go through a classification process and the output is a reactive bound that maps the agent's state x to a set of possible behaviors d for the environment, denoted as $S_d(x)$. To obtain a bound on the influence of x on the environment response d, we want to find a

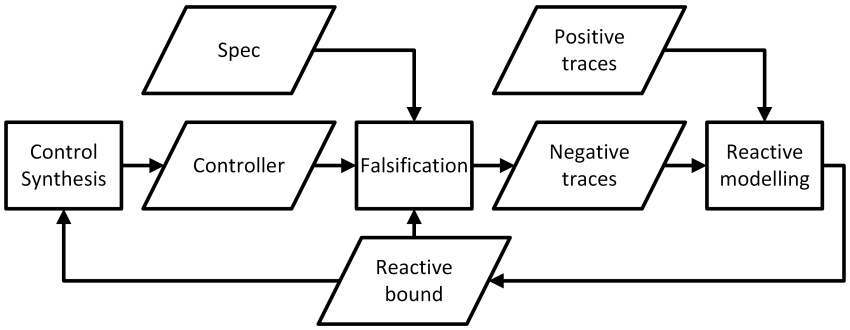


Figure 3.10: Verification with reactive modeling of the environment

function $h : \mathcal{X} \times \mathcal{D} \to \mathbb{R}$ such that $h(x, d) \geq 0$ indicates that d is possible under x, and $h(x, d) < 0$ indicates that d is not possible under x. The set of possible behaviors for the environment given by the reactive bound can be represented as

$$S_d(x) = \{d \mid h(x, d) \geq 0\}. \tag{3.12}$$

Remark 10. Since the data is in the format of snapshots, φ_e is limited to the form $\Box\varphi$ where φ has no dependence on time. One could use a parameterized form for φ_e and project the traces to the parameter space, such as in [117], but this limits φ_e to having only monotonic atoms.

3.10.1 $\mathcal{L}_1$ Piecewise SVM for Reactive Modeling

Given the positive and negative traces, we need to learn an indicator function h, which then gives rise to the reactive bound. This can be solved as a classification problem. There are numerous classification tools in the literature, such as neural networks and logistic regression. For the reactive modeling problem, in addition to good classification accuracy, the following two requirements are critical: i) the probability of false negative should be low, even for the unseen data, and ii) the classification result h should have an analytic form for its classification boundary. For the first requirement, note that $h(x, d) < 0$ indicates that d will not happen under x, and the verification process will ignore such environment input under x. Therefore, the probability of false negative should be very low to guarantee the correctness of the reactive bound and consequently guaranteeing safety. The reliability

analysis of the proposed approach is based on the theory of RCP, which we discuss in detail in Section 3.10.2. For the bound in Lemma 9, we would like the number of parameters for the classifier to be small – this prevents overfitting, and enables us to provide better probabilistic guarantees. The second requirement comes from the fact that the reactive bound will be used explicitly during verification and control synthesis. Therefore, its explicit form should be known.

Due to the two requirements, we choose SVMs with explicit features as the classification method. SVMs fit into the setup of Lemma 9 if the positive data is sampled from an i.i.d. distribution. In particular, we propose an novel expressive $\mathcal{L}_1$ piecewise SVM, which is based on the work on $\mathcal{L}_1$ SVM in [**?**]. [**?**] showed that with a proper cost function, the following optimization solves the $\mathcal{L}_1$ SVM:

$$\begin{aligned} &\min_{v,c,M} k^\mathsf{T} M \\ &\quad \text{s.t. } \|v\| \leq 1,\ y_i = 1 \Rightarrow M_i \geq 0, \\ &\qquad \forall i = 1, ..., N,\ v^\mathsf{T}\phi(z_i)y_i - M_i + cy_i = 0, \end{aligned} \tag{3.13}$$

where $z_i \in \mathbb{R}^m$ is the i-th data point and $\phi : \mathbb{R}^m \to \mathbb{R}^p$ maps the data to the feature space, the flag $y_i = 1$ for positive data points and $y_i = -1$ for negative data points. $v \in \mathbb{R}^p$ is the support vector and c is the offset, therefore the Helly dimension is $p + 2$. $M \in \mathbb{R}^N$ is the slack vector and k is the cost vector with $k_i > 0$ for all i. It is required that all the positive data points are correctly classified (no false negatives), which is needed for the reliability proof.

However, the $\mathcal{L}_1$ SVM suffers from the lack of expressibility, especially for high-dimensional d. We propose two improvements on the SVM: i) Algorithm 4, which generates multiple separating hyperplanes and represents the positive data region as a polytope. ii) Based on Algorithm 4, we introduce Algorithm 5, which allows a piecewise structure for the SVM where a different polytope represents the positive data in each region, and automatically synthesizes the piecewise regions.

The original SVM generates one separating hyperplane in the feature space, which results in a reactive bound with a smooth boundary in $\mathcal{X} \times \mathcal{D}$. In order to make the

reactive bound more expressive, we propose a piecewise $\mathcal{L}_1$ SVM with multiple separating hyperplanes. SVM with multiple separating hyperplanes is achieved with the following greedy algorithm:

Algorithm 4: $\mathcal{L}_1$ SVM with multiple separating hyperplanes

Input: positive data ϕ_p, negative data ϕ_n

$\phi_n^{active} \leftarrow \phi_n$ **for** *i=1:*N_h **do**

  Perform $\mathcal{L}_1$ SVM with ϕ_p and ϕ_n^{active}, get v^i, c^i, slacks M_p, M_n

  $\phi_n^{active} = \left\{\phi_n^{active}(j) | M_n(j) \geq \epsilon\right\}$

end

N_h is the number of separating hyperplanes, ϕ_n^{active} is the set of negative data points that are close to the farthest separating hyperplane, ϵ is the threshold for picking ϕ_n^{active} and M_p and M_n are the slacks for positive and negative data. Each SVM computation generates one hyperplane with v^i and c^i and the indicator function is $h(z) = \min_{i=1,.,N_h} \left\{(v^i)^\intercal z + c^i\right\}$. We can further improve the expressibility by introducing a piecewise structure , which is particularly helpful when the problem itself has a piecewise structure, as demonstrated in Section 4.5.2. Moreover, we develop an auto-tuning piecewise SVM that adjusts the dividing point automatically based on the data by the use of membership functions.

For clarity, we present the piecewise SVM with 2 regions, but note that it can be easily extended to cases with more than 2 regions. Let $g : \mathbb{R}^n \rightarrow \mathbb{R}$ be a scalar function and κ be a scalar variable. We will divide the state space by the threshold $g(z) = \kappa$. First, define the membership functions using the sigmoid:

$$m_1(z, \kappa) = \frac{1}{1 + \exp(\gamma(g(z) - \kappa))}, \; m_2(z, \kappa) = \frac{\exp(\gamma(g(z) - \kappa))}{1 + \exp(\gamma(g(z) - \kappa))},$$

where γ is a tuning parameter that controls the steepness of the sigmoid. Note that $m_1(z, \kappa) + m_2(z, \kappa) = 1$. When there are $d > 2$ regions, one simply constructs d membership functions that are non-negative and add up to 1. With 2 regions, the original feature is extended to $\bar{\phi}(z) = [m_1(z, \kappa) \cdot \phi(z); m_2(z, \kappa) \cdot \phi(z)]$. We then perform $\mathcal{L}_1$ SVM with this new feature vector.

Once the SVM is trained, notice that by equation (3.13), $M_i = v^\intercal \bar{\phi}(z_i) y_i + d y_i$, taking

derivative of the objective function over κ, we have

$$\frac{d(k^\intercal M)}{d\kappa} = \sum_i k_i y_i \frac{d(v^\intercal \bar{\phi}(z_i))}{d\kappa} = \sum_i k_i y_i \left(v_{1:p}{}^\intercal \phi(z_i) \frac{\partial m_1}{\partial \kappa} + v_{p+1:2p}{}^\intercal \phi(z_i) \frac{\partial m_2}{\partial \kappa} \right),$$

$$\left.\frac{\partial m_1}{\partial \kappa}\right|_z = -\left.\frac{\partial m_2}{\partial \kappa}\right|_z = \frac{\gamma m_1(z,\kappa)}{1+\exp(\gamma(g(z)-\kappa))},$$

so we obtain the analytic form of the gradient of the objective function over κ. The overall algorithm alternates between the $\mathcal{L}_1$ SVM and optimizing over κ by gradient descent, as in Algorithm 5.

Remark 11. The setup for $\mathcal{L}_1$ SVM allows for piecewise cost function of M by splitting $M = M^+ + M^-$ with $M^+ \geq 0, M^- \leq 0$: see [**?**] for detail. In the gradient descent step, we maintain the constraint by assigning a large penalty on M^-.

Algorithm 5: Auto-tuning piecewise $\mathcal{L}_1$ SVM

Initialize κ **for** *iter=1:T* **do**

Compute membership functions m_1, m_2

Perform Algorithm 4 with $\bar{\phi}(z) = [m_1(z) \cdot \phi(z); m_2(z) \cdot \phi(z)]$

Perform gradient descent to optimize κ

end

3.10.2 Reliability Analysis with RCP

Next, we provide reliability analysis for the reactive bound. For the ordinary $\mathcal{L}_1$ SVM, we have:

Theorem 12. *Given a positive data set with N points drawn i.i.d. from a fixed (not necessarily known) distribution, and a negative data set, let p be the dimension of the feature vector $\phi(z)$, $p+2 < N$, the $\mathcal{L}_1$ SVM in equation* (3.13) *is always feasible. Denote the solution as $[v, c]$, which satisfies $v^\intercal \phi(z_i) + c \geq 0$, where z_i is the i-th positive data point for $i = 1, 2, ..., N$. Then for an unseen data point z_{N+1} from the same distribution, for $0 < \epsilon < 1$, we have*

$$\mathbb{P}\left\{\mathbb{P}\left\{\neg(v^\intercal \phi(z_{N+1}) + c > 0)\right\} > \epsilon\right\} \leq \sum_{j=0}^{p+2} \epsilon^j (1-\epsilon)^{N-j}. \tag{3.14}$$

Proof. Feasibility can be seen by noticing that the problem is convex and $[w, c, M] = 0$ is a solution. Then note that M can be completely eliminated and represented as a function of $[w, c]$ by the equality constraint, and Helly dimension is upper bounded by $p + 2$ [65]. Thus the upper bound on the probability of misclassification for unseen data is obtained by directly using Theorem 3.3 in [20]: see the proof therein. □

Theorem 12 gives an upper bound for the probability of the probability of misclassification for unseen data to be higher than a threshold, which decreases with the size of the dataset N and increases with the feature dimension p. For SVM with N_h hyperplanes, we provide the following corollary.

Corollary 13. *Given the condition in Theorem 12, the $\mathcal{L}_1$ SVM with N_h separating hyperplanes is always feasible and define $\bar{\epsilon}(k, N) = \min_{0\le\epsilon\le 1} \epsilon + \Phi(\epsilon, k, N)(1-\epsilon)$, where Φ is defined in* (3.11)*, then for any $0 \le \epsilon' \le 1$,*

$$\mathbb{P}\left\{\mathbb{P}\left\{\neg\left(\bigwedge_{i=1}^{N_h}(v^i)^\intercal\phi(z_{N+1}) + c^i > 0\right)\right\} > \epsilon'\right\} \le \frac{\bar{\epsilon}(N, p+2)N_h}{\epsilon'}.$$

Proof. The multi-hyperplane SVM can be viewed as the conjunction of N_h SVMs, therefore we have

$$\mathbb{P}(\bigvee_{i=1}^{N_h}((w^i)^\intercal\phi(z_{N+1}) + c^i < 0)) \le \sum_{i=1}^{N_h}\mathbb{P}((w^i)^\intercal\phi(z_{N+1}) + c^i < 0), \tag{3.15}$$

where for each probability on the right, by Theorem 12, we have $\mathbb{P}(\mathbb{P}((w^i)^\intercal\phi(z_{N+1}) + c^i < 0) > \epsilon) < \Phi(\epsilon, p+2, N)$. Since this is true for all $0 \le \epsilon \le 1$, we have $\mathbb{E}\{\mathbb{P}((w^i)^\intercal\phi(z_{N+1})+c^i < 0)\} \le \min_{0\le\epsilon\le 1} \epsilon + \Phi(\epsilon, p+2, N)(1-\epsilon)$, denoted as $\bar{\epsilon}(N, p+2)$. It is easy to check that this minimum is taken on a bounded function of ϵ on a compact set, therefore the minimum can always be obtained. Then by Markov inequality, we have for $0 \le \epsilon' \le 1$,

$$\begin{aligned}\mathbb{P}\left\{\sum_{i=1}^{N_h}\mathbb{P}((w^i)^\intercal\phi(z_{N+1}) + c^i < 0) \ge \epsilon'\right\} \le& \frac{\sum_{i=1}^{N_h}\mathbb{E}\{\mathbb{P}((w^i)^\intercal\phi(z_{N+1}) + c^i < 0)\}}{\epsilon'} \\ \le& \frac{\bar{\epsilon}(N, p+2)N_h}{\epsilon'}.\end{aligned} \tag{3.16}$$

□

Remark 14. The auto-tuning piecewise SVM in Algorithm 5 changes the optimization problem every time it updates κ, which does not allow us to directly apply Theorem 12. To overcome this, a simple solution is to separate the positive data points into two batches, using the first batch to find a good separation of the state space, i.e., find a good κ, and the second batch to obtain the reactive bound while fixing κ. The size of the second batch determines the probability of misclassification.

3.11 Data-Driven Verification: Case Study

3.11.1 Multi-robot Navigation

As a toy example, we consider a multi-robot navigation problem consisting of two robots as shown in Figure 3.11. We denote the positions of the two robots by $p_1, p_2 \in [-l, l]^2 \subset \mathbb{R}^2$. The robots are characterized by the integrator dynamics $\dot{p}_i = v_i$, where $i \in \{1, 2\}$ and v_i satisfies $\|v_i\|_2 \leq v^{\max}$. The specification for the system is to always maintain distance, i.e., it has to satisfy the specification $\square_{[0,T]}\texttt{connected}$, where T is the time horizon for the STL specification and `connected` is a predicate defined by: $\texttt{connected} \triangleq |p_1 - p_2| \leq r^{\max}$. Here $r^{\max}$ can be thought of as the maximum communication range. The red robot R_1 is the controlled robot and it simultaneously pursues two objects, a moving target T_1 (with bounded velocity), and the blue robot R_2. It follows a given controller, which in simulation is set to be

$$v_1 = \mathbf{sat}_{v^{\max}}(k_1(p_T - p_1) + k_2(p_2 - p_1)), \tag{3.17}$$

with gains k_1 and k_2, where p_T denotes the position of the target and $\mathbf{sat}_a(x) = x$ if $|x| \leq a$ and $a\frac{x}{|x|}$ otherwise. The motion of the blue robot R_2 follows a "black-box" controller, and we would like to learn an overapproximation of its possible behavior as a function of the state. In particular, we pick a controller with a piecewise structure depending on the location R_2:

$$v_2 = \mathbf{sat}_{v^{\max}}\left(\begin{bmatrix} -0.4 & -\beta \\ \beta & -0.4 \end{bmatrix}(p_2 - p_1) + \Delta v_2\right), \tag{3.18}$$

where Δv_2 is a bounded random noise and $\beta = 1$ if $p_{2,1}(t) \leq 0$ and -1 otherwise ($p_{2,1}$ denote the X coordinate of R_2). The above controller roughly makes R_2 spiral counter-clockwise towards R_1 on the left half plane, and spiral clockwise on the right.

To initiate the process, we first collect data with simulation by enforcing equation (3.17) and (3.18) on R_1 and R_2 and let T_1 move randomly in the state space. The positive data collected consists of tuples of $[p_1, p_2, v_2]$. Recall equation (3.12): In the two robot case, the state x is $[p_1, p_2]$, and the environment input d is $[v_2, v_T]$, where $v_T = \dot{p}_T$ is the velocity of the target. We do not explicitly learn a reactive model of v_T and the only constraint for which is the norm bound. In the falsification process, the falsifier can choose v_2 and v_T while v_1 follows equation (3.17). When no reactive bound is in place, the only constraint for v_2 is the norm bound $v^{\max}$, and the falsifier can easily find falsifying traces. The falsifying traces then generate negative data points with the same structure as the positive data, which is then fed to the reactive modeling module. The reactive modeling module utilizes the auto-tuning piecewise $\mathcal{L}_1$ SVM algorithm introduced in Section 3.10.1 with 3 separating hyperplanes and $g(x) = p_{2,1}$ to construct a reactive bound. We choose features that are linear in v_2 so that the resulting reactive bound is a polytope $S_{v_2}([p_1, p_2]) \in \mathcal{R}^2$, given $[p_1, p_2]$. The reactive bound is then fed to the falsifier, which would project the raw input of v_2 to $S_{v_2}([p_1, p_2])$. Since we construct the features for the SVM such that S_{v_2} is a polytope, the projection is easily solved with quadratic programming.

After 5 iterations of updating the reactive bound, the falsifier cannot find a falsifying trace, which means that the controller is verified under the learned reactive bound. Moreover, it turns out that the threshold κ converges to 3×10^{-3}, which is very close to the actual threshold at $\kappa = 0$.

Figure 3.11 shows the two robot positions with corresponding reactive bounds. When R_2 is on the right, the reactive bound allows it to spiral clockwise, while the direction of spiraling flips on the left side. But importantly, the worst-case v_2, which is to move away from R_1 with $v^{\max}$ is not allowed in both cases. For this example, we use $40,000$ snapshots with 45 features for the SVM. For $\epsilon = 0.01$, $\mathbb{P}(\mathbb{P}(\textit{misclassification}) \geq \epsilon) < 0.007$ by Corollary 13.

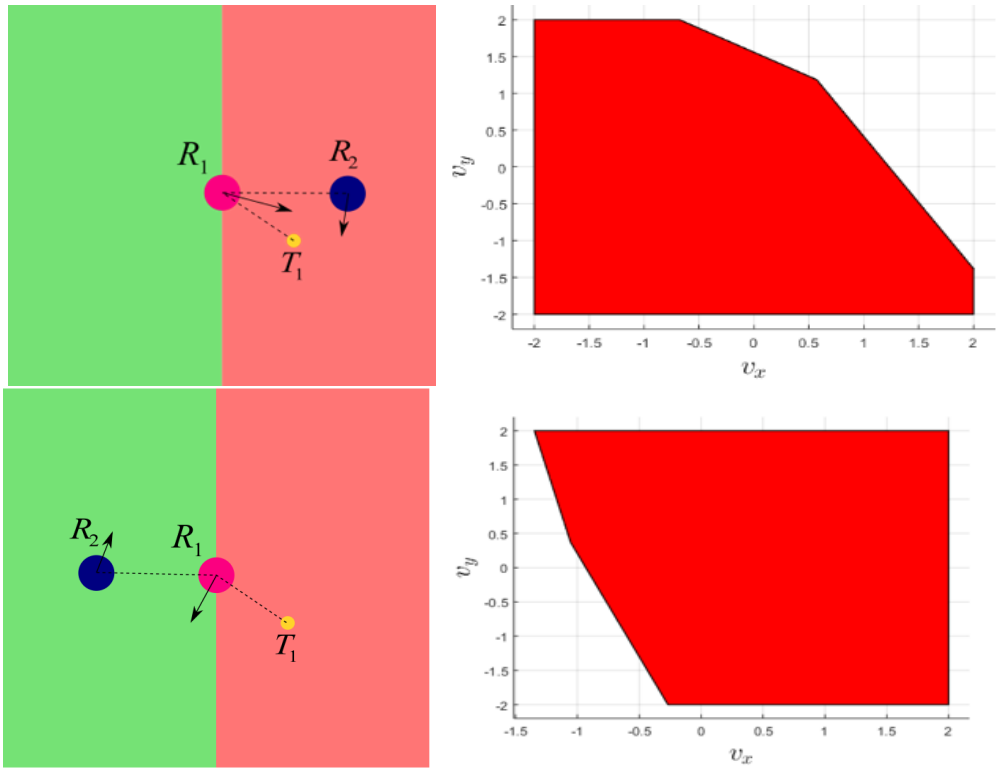


Figure 3.11: Two robot scenario and Reactive bounds.

3.11.2 Lane Change

A practical application of the proposed method is verification of the lane-change control for autonomous driving. We would like to guarantee with high probability that a given controller can safely finish a lane change within a given horizon. We consider a scenario as depicted in Figure 3.12, where the autonomous vehicle (AV) attempts to make a lane change with the human-driven vehicle (HV) on the back. The state of the system is

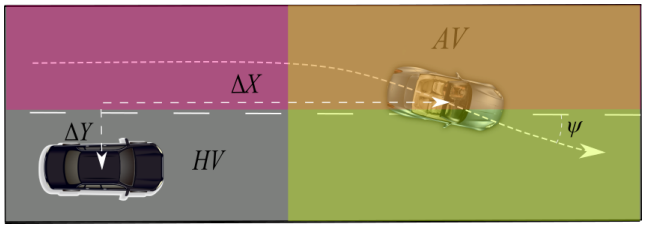


Figure 3.12: Lane-Change Scenario

$x = \begin{bmatrix} \Delta X & \Delta Y & \Delta v & \psi \end{bmatrix}^\mathsf{T}$, where ΔX and ΔY are the longitudinal and lateral coordinate differences between the two vehicles, Δv is the velocity difference and ψ is the heading

angle of the AV. The input of AV are the acceleration a_1 and yaw rate r_1, and the input of the HV is the acceleration a_2. The dynamics is given by

$$\dot{x} = \begin{bmatrix} \Delta v & v_1 sin(\psi) & a_1 - a_2 & r_1 \end{bmatrix}^\mathsf{T}. \tag{3.19}$$

The specification for the problem is to always not collide and keep within the lane, and eventually finish the lane change within horizon T. Formally, the specification is expressed in STL:

$$\square_{[0,T]}(\neg \mathbf{COL} \wedge \mathbf{LK}) \wedge \Diamond_{[0,T]} \mathbf{LC}, \tag{3.20}$$

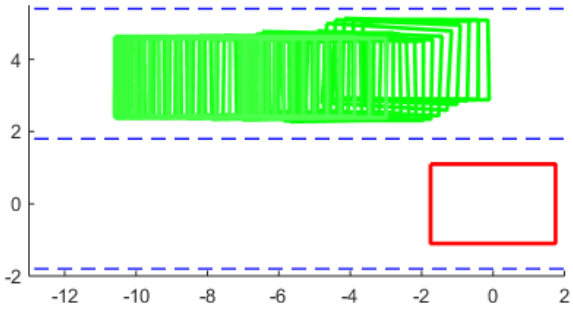
where **COL** stands for collision, **LC** stands for lane change and **LK** stands for lane keeping, which all can be represented as subsets of the state space:

$$\begin{aligned} \mathbf{COL} \Leftrightarrow \quad & |\Delta Y| \leq a \wedge |\Delta X| \leq b \\ \mathbf{LC} \Leftrightarrow \quad & |\Delta Y| \leq \epsilon, \\ \mathbf{LK} \Leftrightarrow \quad & 0.5w - 0.5b \geq \Delta Y \geq -1.5w - 0.5b, \end{aligned} \tag{3.21}$$

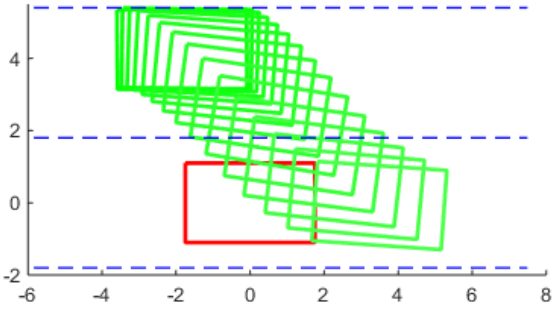
where a, b are the length and width of a typical car, w is the width of a lane and $\epsilon \in \mathbb{R}_+$ is a small constant. As an example, we consider a model-predictive control scheme with mixed integer programming as the controller for the AV. As shown in Figure 3.12, the AV should stay within the union of the two colored regions within the prediction horizon T, which is enforced by the "big M" procedure as a mixed integer linear constraint. The MPC controller takes the current value of a_2 and assumes an exponential decay within the prediction horizon $a_2(t) = a_2(0)e^{-t/\tau}$. This is not accurate but a heuristic prediction used to plan for the controller. The lane keeping constraint is also enforced as a linear constraint and the objective function penalizes ΔY, driving the vehicle to finish the lane change.

The lane-change problem was studied in [**?**], and we use the same source for positive data, which is from the safety pilot model deployment (SPMD) database with more than 50 million miles of naturalistic driving data [14]. The feature structure is also inherited from [**?**]. Following the procedure shown in Figure 3.10, the falsification tool starts with simply the physical limit of a_2 and tries to falsify the specification in equation (3.20), the falsifying

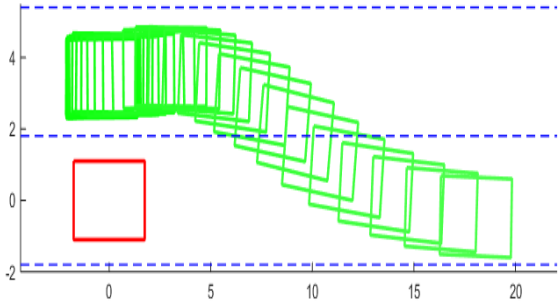
traces are then broken into snapshots and treated as negative data. The SVM procedure then generates the reactive bound for a_2. In the lane-change case, it is not difficult to see that the safety specification is monotonic w.r.t. a_2, i.e., it is always safer for the HV to decelerate. Therefore, the reactive bound for a_2 is in the form of an upper bound $a^{\star}_{\max}(x)$ that changes with the state x.



(a) Falsify by blocking



(b) Falsify by collision



(c) Success run

Figure 3.13: Verification of lane change

The result of verification for the MPC controller is shown in Figure 3.13. Without the reactive bound, the falsification procedure is able to falsify the specification by accelerating

and blocking the AV from finishing a lane change, as shown in Figure 3.13a, and the verification procedure terminates after a maximum iteration number. After 4 iterations, the SVM presented in Section 3.10.1 generates a reactive bound that makes the falsification infeasible, i.e., verifies that the MPC controller satisfies the specification and a success run is shown in Figure 3.13c. However, when we remove the collision avoidance constraint in the MPC controller, the falsification tool finds a falsifying trace by causing a collision with the AV (Figure 3.13b), thereby providing feedback for the controller design process.

Run-time considerations The mean run times over 5 runs are 4977 s and 29.6 s for the robot problem and for the lane-change problem, respectively.

Remark 15. Not every snapshot from the falsifying trace is included in the negative data. We use an *ad hoc* selection scheme to pick out 'important' snapshots based on criterion such as the distance between the robots (Section 4.5.2), and lateral position for the AV (Section 3.11.2).

3.12 Discussion: Data-driven Verification

In Section 3.10 we presented a framework that combines falsification and specification learning to learn an overapproximation of the reactive behavior of the environment from real data. There are two key parts of the algorithm, the falsifier and the reactive modeling module. The falsifier can handle specifications written in temporal logic and generate falsifying traces, which is then used by the reactive modeling module together with the positive data to generate the reactive bound. The reliability of the reactive bound is guaranteed by the theory of RCP, which can give probabilistic guarantees determined by the amount of data available. In Section 3.11, we showed the capability of the proposed framework to handle nonlinear environment behavior. We also demonstrated the applicability of the approach to a practical problem in autonomous driving with real-world human-driving data. The framework presented here provides a general approach for the diagnosis of autonomous agents interacting with complex environments, such as in the case of human-robot interaction.

Chapter 4

Verifying, Interpreting and Debugging Learned Systems

While neural networks have demonstrated significant potential as key building blocks of intelligent systems, and have shown immense progress on diverse tasks [91, 105, 109], they are often deployed without formal guarantees of their correctness and functionality. However, interpreting the behaviors of complex neural networks and characterizing their exact behaviors is an extremely difficult task, which poses a major challenge to their use in safety-critical systems. Their performance is typically evaluated using test data, or sometimes with adversarial evaluation [21, 45, 114, 119]. However, such evaluation does not provide formal guarantees regarding the absence of rare but possibly catastrophic failures [1, 99, 115]. Another related direction is the formal analysis of neural networks, where there has been significant progress on developing faster algorithms for validating or falsifying formal properties of whole neural networks directly through their encoding as constraint satisfaction problems [19, 75, 125]. Along this direction, most of the focus in this direction has been restricted to feedforward networks and robustness to adversarial perturbations [78, 98, 112]. A recent direction for training reliable neural networks that has shown promise is that of folding the verification procedure into the training loop [59, 90]. Following training, this allows us to provide formal guarantees using tractable algorithms in a scalable manner.

We build on the aforementioned progress in evaluating and designing reliable neural networks by making two key contributions:

- First, we propose an alternative approach for analysis of neural networks that involves decomposing the large networks into modules, in contrast with the more commonly studied approaches that perform a monolithic analysis on the neural network [19, 75, 125]. Such monolithic analysis often does not provide a lot of insight into the functioning of the neural networks. Performing modular analysis is a standard practice in software program analysis [67]. A crucial component for enabling such modular analysis is that we must be able to represent and manipulate *pre-images* of programs, or computable functions in general. On neural networks, this translates to being able to propagate sets of the network outputs backwards through individual neural layers (as real-valued functions), eventually to the input domain. However, this is in principle harder than direct constraint solving, because of the requirement of representing and manipulating high-dimensional geometric shapes that often do not have polynomial-size representations. Thus, the important question is how to efficiently compute approximate representations (abstractions) of such pre-image sets, so that they are both compact and precise enough for enabling formal analysis, interpretation and knowledge/policy extraction.

  To answer this question, in the first part of this chapter we develop algorithms for computing *symbolic abstractions* of pre-images of neural networks. We bypass the difficulty of representing the exact pre-images by maintaining both overapproximations and underapproximations that can be compactly represented as symbolic constraints. We leverage a recent algorithm for computing symbolic *interpolants* [2], where an extension of Farkas' lemma is used to *learn* interpolants that have simple structures. The techniques are applicable because the concepts that are learned by neural networks are often simple [7]. We exploit the network structures and propagate pre-images of subsets of the output space through each layer to the input space. We enhance scalability of the algorithms on piecewise-linear neural networks by designing heuristics for the specific symbolic forms of the abstractions.

  Through experiments for two control environments, a cart-pole system and a swimmer model, we demonstrate the applicability of the approach for knowledge/policy

analysis and extraction. We show that for the multilayer perceptron (MLP) network policies trained through standard reinforcement learning algorithms, we can extract knowledge in the form of compact abstractions. For the cart-pole system, the extracted policy achieves a perfect score. Using the extracted policy we are able to formally verify/falsify certain complex safety properties. For the swimmer model, we show how high-torque outputs are mapped to a compact representation in the input space.

- Second, we propose extensions to recent work [59, 90], which is based on propagating differentiable numerical bounds through deep neural networks (DNNs), to include temporal specifications that go beyond adversarial robustness and consider novel auto-regressive architectures such as gated recurrent units (GRUs) [25]/ recurrent neural networks (RNNs). This is important as many practically relevant systems involve DNNs that lead to sequential outputs (e.g., an RNN that generates captions for images, or the states of a reinforcement learning (RL) agent), and there are many properties of interest that go beyond simple input-output robustness. To handle the auto-regressive decoder often used in RNN-based systems, we leverage differentiable approximations of the non-differentiable operations during training.

  We also empirically demonstrate the applicability of our approach to ensure verifiable consistency with temporal specifications while maintaining the ability of neural networks to achieve high accuracy on the underlying tasks across domains. For supervised learning, verified training on the training data enables us to provide similar verification guarantees for unseen test data. We find that verified training results in robust DNNs whose specification conformance is significantly easier to guarantee than those trained adversarially or with data augmentation.

4.1 Preliminaries

4.1.1 Neural Networks as Constraints

Consider a neural network f with n layers. That is, $f(x) = h.g_n.g_{n-1}.\ldots.g_1(x)$ where g_r is the transfer function representing the map from the input to the output space for layer r and

$h : \mathbb{R}^k \rightarrow Y$ is a map from the logits to the k class-labels (e.g., arg max). For example, if a network has $n-1$ layers with ReLU activation and a final linear layer:

$$g_r(z) = \max(W_r x + b_r, 0) \qquad \forall r \in 1, 2, \ldots, n-1$$
$$g_r(z) = W_r x + b_r \qquad r = n.$$

For simplicity, we only discuss neural networks that map to a discrete set of outputs, but the approach is valid even when the network produces a continuous set of outputs. By $y_i^f(x)$, we refer to the output from the i^{th} layer of the neural network, i.e., $y_i^f(x) = g_i.g_{i-1}.\ldots.g_1(x)$. Often, in classification tasks, the outputs from the layer g_n are fed through a softmax layer to normalize the scores.

For the first half of this chapter, we restrict ourselves to networks that can be expressed in (quantifier-free linear rational arithmetic) QFLRA. A large class of neural networks, namely piecewise linear neural networks (without the softmax layer) can be expressed as constraints in the theory of QFLRA. This includes neural networks with activation functions that are piecewise-linear (e.g., ReLU, Leaky ReLU, MaxOut, MaxPool). We follow the encoding described in [46]. For example, a ReLU node $y = \max(0, x)$ is written as

$$(y = 0 \wedge x \leq 0) \vee (y = x \wedge x \geq 0).$$

The entire network is similarly encoded into QFLRA.

As a slight abuse of notation we interchangeably use $g(z)$ to represent the map of z through the function g, and the first-order logic constraint that enforces the same. For example, consider the function $g(x) = \max(0, w^T x)$. We interchangeably use $y = g(x)$ to also represent the constraint $(y = w^T x \wedge w^T x \geq 0) \vee (y = 0 \wedge w^T x < 0)$. For a satisfying assignment (y, x) for this constraint, we have $y = g(x)$. For a formula φ that has a vector of free variables s, we write $x \models \varphi$ if φ interprets to `True` when we set $s = x$.

4.1.2 Symbolic Interpolants

Symbolic interpolation is a well-studied concept in propositional and first-order logic [30]. Given two quantifier-free first-order formulas A and B, such that $A \wedge B$ is unsatisfiable, a Craig interpolant I is a formula satisfying

- $A \implies I$;
- $B \wedge I \implies \texttt{False}$;
- I only contain variables that are shared by A and B.

Intuitively, the interpolant I provides an overapproximation of A that is still precise enough to exhibit its conflict with B, and does not contain redundant information that involves any variable that is not shared by both A and B. When $A \wedge B$ is not satisfiable, Craig's interpolation theorem guarantees the existence of an interpolant I such that I overapproximates A and $\neg I$ overapproximates B. These interpolants have found application in compositional approaches to program verification and SMT solving. In our work, we build on the algorithm from [2] for computing interpolants, as opposed to other approaches based on lazy SMT that produces complex interpolants. The intuition behind this choice is that simpler interpolants are more likely to provide general explanations corresponding to the neural network's parameters and the task for which the network was trained, rather than complex ones that may overfit the specific instantiation. Further, simpler interpolants provide the added advantage of being easier to reason over using automated reasoning engines (e.g., `z3`).

4.2 Computing Pre-Image Abstractions

Here, we outline an algorithm for computing compact abstractions for the *pre-images* for piecewise linear neural networks. We first begin by defining pre-images for a neural network f.

Definition 16 (Pre-images)**.** *Consider a neural network f. Let X be the domain and Y*

be the codomain. The pre-image of a set $S \subset Y$ for the neural network f is the set $\{x \in X | f(x) \in S\}$.

For example, consider a neural network-based cart-pole controller with the action space $\{\texttt{left}, \texttt{right}\}$. The pre-image corresponding to $S = \{\texttt{right}\}$ is the set of observations that cause the controller to output `right` as the action. In the remainder of this chapter, we refer to this pre-image of set S for the neural network f as $\text{Pre}_f(S)$. The exact pre-image of the network for a given set S, in the worst case, can have exponentially many linear regions. To overcome this we consider abstractions that provably (over) underapproximate the exact pre-image $\text{Pre}_f(S)$. The restriction on the structure of S here is that it has to be expressible in QFLRA, which includes all constraints that have half-spaces as atoms combined with Boolean operators. Here, in our analysis, we do not consider the softmax layer as it preserves the ordering amongst scores corresponding to the different classes. For a vector $z \in \mathbb{R}^m$, $\arg\max_i\{z_i\} = \arg\max_i\{\texttt{softmax}(z)_i\}$.

We seek to compute approximations for the pre-image that closely approximate the pre-image, but are provable (super) subsets of the pre-image. The fact that these are provable (over) underapproximations (unlike the approximated models in [13, 118]) allows us to prove properties that hold for the neural network itself. For example, suppose we wish to prove a property that for every input from some set W, the corresponding output from the neural network does not belong to the set S. By computing an overapproximation O for $\text{Pre}_f(S)$ and showing that $O \wedge W$ is not satisfiable we have verified the property. Similarly, if the property was that every output belongs to some set S, then by computing the underapproximation U for $\text{Pre}_f(S)$ and showing that $W \implies U$ is valid, we have verified the property.

Here, we give a brief overview of the algorithm for computing the overapproximation of the pre-image for set S. Consider the neural network described earlier. Let $p_n^{(f,S)}$ be the set of inputs to layer g_n of the neural network that lead to the output being in set S. Similarly, let $p_{n-1}^{(f,S)}$ be the set of inputs to layer g_{n-1} of the neural network that result in outputs in S. Note that $p_n^{(f,k)}$ is the set of assignments to s that satisfy

$$h(g_n(s)) \models S.$$

For the other layers ($r < n$), we can iteratively define $p_r^{(f,S)}$ as assignments to s that satisfy

$$p_r^{(f,S)} = \{s \mid g_r(s) \models p_{r+1}^{(f,S)}\}. \tag{4.1}$$

The core idea is to begin by computing an approximate representation of $p_n^{(f,k)}$, and using this to then compute the approximations for $p_{n-1}^{(f,k)}$. Then, by iterating through the layers of the network, we can compute an approximation for $\text{Pre}_f(S)$. Computing these approximations involves proving the interpolation condition (see Section 4.2.1). We could compute the approximation across the entire network, but proving the interpolation condition for the entire network is computationally expensive. This is because the worst-case complexity of proving properties for piecewise-linear networks scales exponentially in the number of nodes under consideration [75]. The layer-wise approach breaks down the problem, where we compute approximations for each layer. This requires proving properties across one layer at a time and can be further simplified to computing approximations over sets of nodes instead of entire layers.

4.2.1 Computing Overapproximations

Here we outline how to compute a useful overapproximation of $p_r^{(f,S)}$, assuming we have the overapproximation of $p_{r+1}^{(f,S)}$. Denote the overapproximation of $p_{r+1}^{(f,S)}$ as $O_{r+1}^{(f,S)}$ with $O_{n+1}^{(f,S)} = S$. First, consider a set of randomly chosen points $\bar{X}$, either by sampling from the input domain or from the training data. Let $X_S \subseteq \bar{X}$ be the set of points such that for every $x \in X_S$, $f(x) \notin S$. Introduce the auxiliary free variable vector $\bar{p}_r$ and construct the formula

$$\phi_{r-1} := \bigvee_{\bar{x} \in X_S} \left(\bar{p}_r = y_{r-1}^f(\bar{x}) \right). \tag{4.2}$$

This formula allows $\bar{p}_r$ to assume the value of the output at layer g_{r-1} corresponding to inputs from X_S. Recall that $y_{r-1}^f(x) = g_{r-1}.g_{r-2} \ldots g_1(x)$, i.e., $y_{r-1}^f(x)$ is the vector of activation values corresponding to x from layer $r-1$ of the network. Now, consider the formula

$$\xi_r := g_r(\bar{p}_r) \models O_{r+1}^{(f,S)}. \tag{4.3}$$

Note that the set of valid assignments for $\bar{p}_r$ represents an overapproximation of the set $p_r^{(f,S)}$. This is because

$$\left(g_r\left(\bar{p}_r\right) \models p_{r+1}^{(f,S)}\right) \implies \left(g_r\left(\bar{p}_r\right) \models O_{r+1}^{(f,S)}\right),$$

which follows from $O_{r+1}^{(f,S)}$ overapproximating $p_{r+1}^{(f,S)}$ and as a consequence of equation (4.1).

Lemma 17. *If $O_{r+1}^{(f,S)} \wedge \phi_r$ is unsatisfiable, then the formula $\xi_r \wedge \phi_{r-1}$ is unsatisfiable for each $r \in \{1, 2, \ldots, n\}$.*

Proof. Assume $O_{r+1}^{(f,S)} \wedge \phi_r$ is unsatisfiable. Suppose $\exists \bar{p}_r$ satisfying ϕ_{r-1} and ξ_r. By definition of ϕ_r, we have $g_r(\bar{p}_r) \models \phi_r$ and by equation (4.3), $g_r(\bar{p}_r) \models O_{r+1}^{(f,S)}$. This results in a contradiction since $O_{r+1}^{(f,S)} \wedge \phi_r$ is unsatisfiable. □

Lemma 18. *If $O_{r+1}^{(f,S)} \wedge \phi_r$ is unsatisfiable, $\exists I_r$ that satisfies the following:*

$$p_r^{(f,S)} \implies \xi_r \implies I_r, \tag{4.4}$$

$$I_r \implies \neg\phi_{r-1}. \tag{4.5}$$

Proof. Lemma 17 and Craig's Interpolation theorem guarantee the existence of an I_r satisfying the two conditions. □

Lemma 18 guarantees the existence of an overapproximator of $p_r^{(f,S)}$ that is disjoint from the set of sampled points that map to the complement of S. This ensures that the overapproximations do not get arbitrarily slack as we propagate the interpolants through the layers and remain tight. Further, since we use the algorithm from [2] that is designed to compute simple interpolants, they are likely to generalize to other data points. A formal description is provided in Algorithm 7. Note that I_r is only a function of the $\bar{p}_r$ (the only shared free variables between ϕ_{r-1} and ξ_r). Figure 4.1 outlines the setup for a single layer.

4.2.2 Bounding the Problem

The interpolants computed that serve as overapproximations are in the disjunctive normal form (DNF) with the atoms being half-spaces. The convergence of the algorithm for

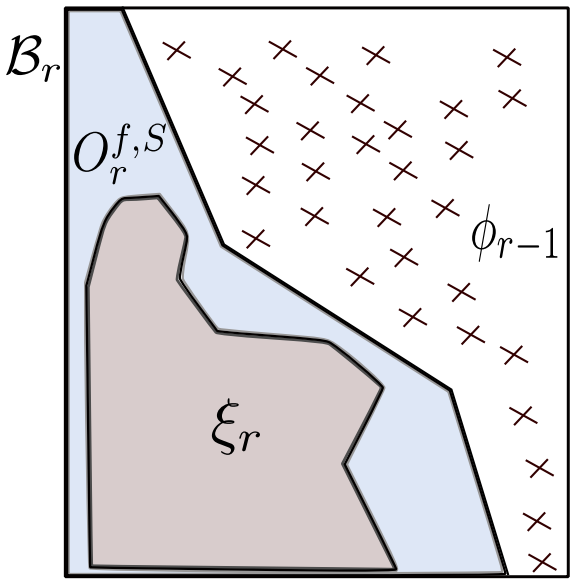


Figure 4.1: Illustration of the approach. ξ_r is the set of inputs to layer $g_r(.)$ that map into the overapproximating abstraction for the subsequent layer $g_{r+1}(.)$. ϕ_{r-1} is a set of inputs to layer $g_r(.)$ sampled so that the neural network maps them to outside the set S, where S is the set whose pre-image is being abstracted.

computing the interpolant can be made faster by restricting the domain in which ϕ_{r-1} and ξ_r need to be separated by the interpolant. Given a lower bound (l_r) and an upper bound (u_r) on the outputs y_r^f, we can construct an additional constraint $\mathcal{B}_r$ defined as

$$\mathcal{B}_r := (\bar{p}_r \leq u_r) \wedge (\bar{p}_r \geq l_r).$$

We then compute the interpolant such that $(\mathcal{B}_r \wedge \xi_r) \implies I_r$ and $I_r \implies \neg\phi_{r-1}$. To compute u_r and l_r we use the relaxation proposed in [46]. In most tasks, the inputs come from a bounded domain. For example in image processing, pixels have values ranging between 0 and 255. These bounds can then be propagated through the network as in [46] using a convex relaxation of the network. Every ReLU node $y = \max(0, x)$ that behaves non-linearly is approximated with its convex hull:

$$y \geq 0, y \geq 0, y \leq u_x \frac{x - l_x}{u_x - l_x}. \tag{4.6}$$

Using this relaxation for the nonlinear constraints results in a linear program (LP). By optimizing over the resulting LP, the bounds for the nodes can be computed in a sequential manner, starting with the input layer. These bounds can further be tightened by splitting the input domain as in [19], but for our work we do not split the input domain. As an added benefit, bounding the problem makes checking the interpolation condition significantly faster.

Algorithm 6: Computing compact abstractions

Output Returns Simple Overapproximator of Pre(S)
:
Compute $\forall \bar{x} \in X_S$, $y_1^f(\bar{x}) = g_1(\bar{x})$
Compute l_1, u_1 and $\mathcal{B}_1$
for r=$2 \ldots n$ **do**
    Compute $\forall \bar{x} \in X_S$, $y_r^f(\bar{x}) = g_r(\bar{x})$
    Compute l_r, u_r and construct $\mathcal{B}_r$
end
for r=$n \ldots 1$ **do**
    Construct ϕ_{r-1} (See equation (4.2))
    Construct $\xi_r \wedge \mathcal{B}_r$ (See equation (4.3))
    Compute I_r satisfying equations (4.4) and (4.5)
    Set $O_r^{(f,S)} = I_r$
end
return $O_1^{(f,S)}$

Theorem 19. *(Soundness and Completeness) Algorithm 7 always terminates to return an overapproximator* $O_1^{(f,S)}$ *of* Pre(S)*. Further,* $O_1^{(f,S)}$ *is disjoint from* X_S*.*

Proof. At $r = n$, by construction we have $O_{r+1}^{(f,S)} \wedge \phi_r$ is not satisfiable. Lemmas 17 and 18 guarantee the existence of an overapproximator (interpolant) satisfying equations (4.4) and (4.5). The algorithm from [2] is both sound and complete, and hence is guaranteed to find an interpolant if one exists. For $r = n - 1$, we again have that $O_{r+1}^{(f,S)} \wedge \phi_r$ is not satisfiable since $O_{r+1}^{(f,S)} \implies \neg\phi_r$ (equation (4.5)). Repeating the arguments above, for every $r \leq n$, we compute $O_{r+1}^{(f,S)}$ (I_r) satisfying equations (4.4) and (4.5). Hence, Algorithm 7 terminates to return an overapproximator for Pre(S) (equation (4.4)) that is disjoint from X_S (equation (4.5)). □

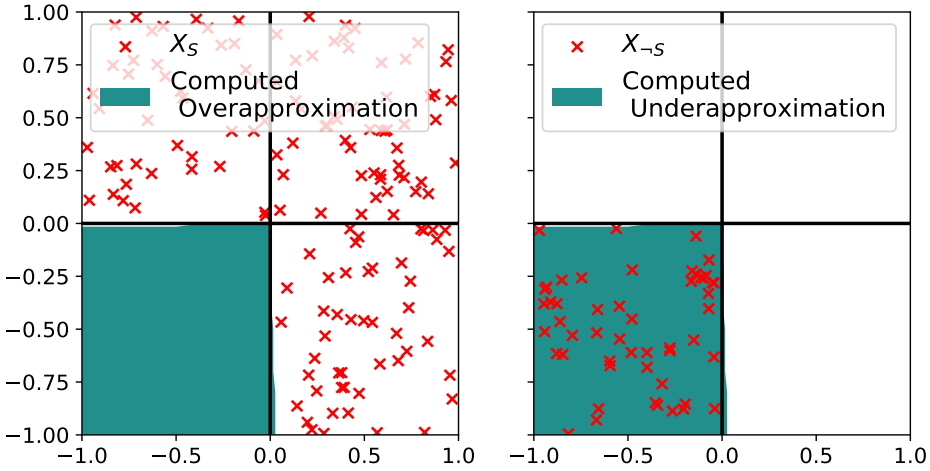


Figure 4.2: Left: An overapproximation of the region classified by the network as the third quadrant. Right: An underapproximation of the region classified by the network as the third quadrant. The overapproximation and the underapproximation are close to each other. It can be observed that in both the under and the overapproximations, parts of the third and the fourth quadrant are misclassified – this indicates faulty behavior for the neural network f.

4.2.3 2D Example

To illustrate the ideas developed above, we introduce a small example. Consider a simple-neural network $f : \mathbb{R}^2 \rightarrow \mathbb{R}^4$, with two hidden layers with 10 ReLU nodes in each layer. The network is trained to predict which quadrant a point x belongs to and achieves an accuracy of 99.65% on hold-out data. Here, $\arg\max(f(x))$ represents the quadrant x belongs to. Along each dimension, the input is restricted to the domain $[-1, 1]$. Here, we are interested in computing a compact representation of the set the network classifies as the third quadrant. To compute an overapproximation, we sample a set of 150 points that are classified as being outside the third quadrant by the neural network. Here, $O_n^{f,S} = \bigwedge_{j=1,2,4} (y_3 > y_j)$ where $y = f(x)$ is the output of the neural network. To compute the underapproximation, we sample 50 points that are classified as being in the third quadrant. Here, $O_n^{f,S} = \bigvee_{j=2,3,4} (y_3 < y_j)$ and Figure 4.2 depicts the over and underapproximations computed. The overapproximation computed is a union of 5 polytopes while the underapproximation computed consists of 2 polytopes.

4.3 Computing Pre-Image Abstractions: Algorithms

The core of the computational effort in Algorithm 7 comes from Line 10 where the interpolant I_r is computed. Computing the interpolant for two constraints A and B has the following key steps:

- Sampling polytopes satisfying A and B and separating them with Farkas' lemma.
- Checking the conditions $A \implies I$ and $I \implies \neg B$, and generating counter examples (if any).
- Merging and splitting of the polytopes into sets, which are then separated by Farkas' lemma.

Counterexample-guided abstraction refinement (CEGAR) [29] – checking the interpolation condition and generating counterexamples to refine the abstraction – is integral to computing the interpolants. The sampling at the first stage of CEGAR can done with training samples, and subsequently an oracle can be used to find counterexamples to the condition $A \implies I$ and $I \implies \neg B$. Checking the conditions $A \implies I$ and $I \implies \neg B$ using an external oracle turns out to be the most expensive part of the algorithm.

Note that in our work A encodes the behavior of a layer of the neural network (with the nonlinearity), and conventional SMT solvers are inefficient at verifying properties for nonlinear neural networks. For checking $A \implies I$, we use the framework *PLNN-v* from [19] designed for verifying piecewise-linear neural networks. PLNN-v utilizes an encoding where the network and the property to be verified are encoded as a single network ($\bar{f}$), and an optimization problem is solved to determine if there exists an input to generate an output whose value is greater than 0. If there is none, the property is `unsat` over the input domain for original neural network f. $A \implies I$ is verified by checking that $A \wedge \neg I$ is unsatisfiable. Recall that for each r, $O_r^{f,S}$ is in DNF with linear atoms. For a layer $g_r(.)$, A has the form

$$(s = g_r(\bar{p}_r)) \wedge \left(\bigvee_i T_i s \leq t_i \right)$$

and I has the form $\left(\bigvee_i Q_m \bar{p}_r \leq q_m\right)$. Then, $A \wedge \neg I$ can equivalently be written as

$$\begin{aligned}(s = g_r(\bar{p}_r))) \wedge (-\max(\max_i(-\max_j(-Q_{i,j}\bar{p}_r + q_{i,j}),\\ -\max_m(-\max_n(-T_{m,n}s + t_r)) \geq 0).\end{aligned} \tag{4.7}$$

This can then be encoded into a neural network with $g_r(.)$ as the first layer followed by a sequence of MaxPool layers to encode the above constraint. For checking $I \implies \neg B$ we use the SMT solver z3 [37].

Splitting Heuristic The algorithm for computing interpolants from [2] relies on sampling and separating sets of polytopes S_A and S_B satisfying formulas A and B, respectively. The objective is to separate the sets of polytopes by sequentially applying a set of merging and splitting heuristics with hyperplanes generated using Farkas' lemma. If the two sets of polytopes cannot be separated by a single hyperplane, the sets are broken down into smaller subsets using the `unsat-core` returned by z3. However, generating the `unsat-core` is computationally expensive and further, we do not use $z3$ for checking $A \implies I$. Alternatively, we develop an intuitive heuristic where we first consider one polytope each satisfying $A_i \in S_A$ and $B_i \in S_B$. Then, we compute a separating hyperplane $\langle w, x\rangle + b = 0$ such that $\langle w, x\rangle + b < 0 \implies A_i$ and $\langle w, x\rangle + b > 0 \implies B_i$. Subsequently, we check if any of the other polytopes in our set are already separated by this hyperplane. If there exists a polytope p satisfying A that is not separated by the hyperplane, we combine p with the rest of the separated polytopes from A and compute a new hyperplane $\langle \hat{w}, x\rangle + b = 0$. If we cannot compute such a hyperplane, we split p from the rest. This is outlined in Algorithm 7, and the polytopes returned by Algorithm 7 are split from the initial set. In Algorithm 7, x is the set of shared free variables for formulas A and B.

Merging Heuristic In [2], the heuristic used for merging is to group together polytopes based on the syntactic similarity. However, for our problem all the sampled polytopes corresponding to the generated counterexamples during CEGAR are syntactically similar. Instead, we merge the polytopes into the set that has the closest matching pattern of activation

Algorithm 7: Splitting Heuristic

Input : $S_A = \{A_1, \ldots, A_c\}$ (Polytopes satisfying A), $S_B = \{B_1, \ldots, B_d\}$ (Polytopes satisfying B)

Output Set of polytopes to be split from S_A and S_B.

:

Set $A_{\text{count}} = 1$,$B_{\text{count}} = 1$
Compute (w, b):$\langle w, x\rangle + b = 0$ separates A_1, B_1
$A_{\text{sat-set}} = \emptyset$, $B_{\text{sat-set}} = \emptyset$, Unsat-Set = $\emptyset$
while $A_{count} \leq c \vee B_{count} \leq d$ **do**
    $A_{\text{old-count}} = A_{\text{count}}$, $B_{\text{old-count}} = B_{\text{count}}$
    for $i = A_{old\text{-}count}, \ldots, c$ **do**
        $A_{\text{count}} = A_{\text{count}} + 1$
        if $A_i \notin A_{sat\text{-}set}$ **then**
            if $\langle w, x\rangle + b < 0 \implies A$ **then**
                $A_{sat-set} = A_{sat-set} \cup \{A_i\}$
            else
                $\bar{S}_A = A_{sat-set} \cup \{A_i\}$
            end
            try:
                Find (w, b) to sep. $\bar{S}_A$, $B_{\text{sat-set}}$
            catch:
                Unsat-Set = Unsat-Set $\cup \{A_i\}$
                break
            end
        end
    end
    for $i = B_{old\text{-}count}, \ldots, d$ **do**
        $B_{\text{count}} = B_{\text{count}} + 1$
        if $B_i \notin A_{sat\text{-}set}$ **then**
            if $\langle w, x\rangle + b > 0 \implies B$ **then**
                $B_{sat-set} = B_{sat-set} \cup \{B_i\}$
            else
                $\bar{S}_B = B_{sat-set} \cup \{B_i\}$
                try:
                    Find (w, b) to sep. $\bar{S}_B$, $A_{\text{sat-set}}$
                catch:
                    Unsat-Set = Unsat-Set $\cup$ $\{B_i\}$
                    break
                end
            end
        end
    end
end
return Unsat-Set

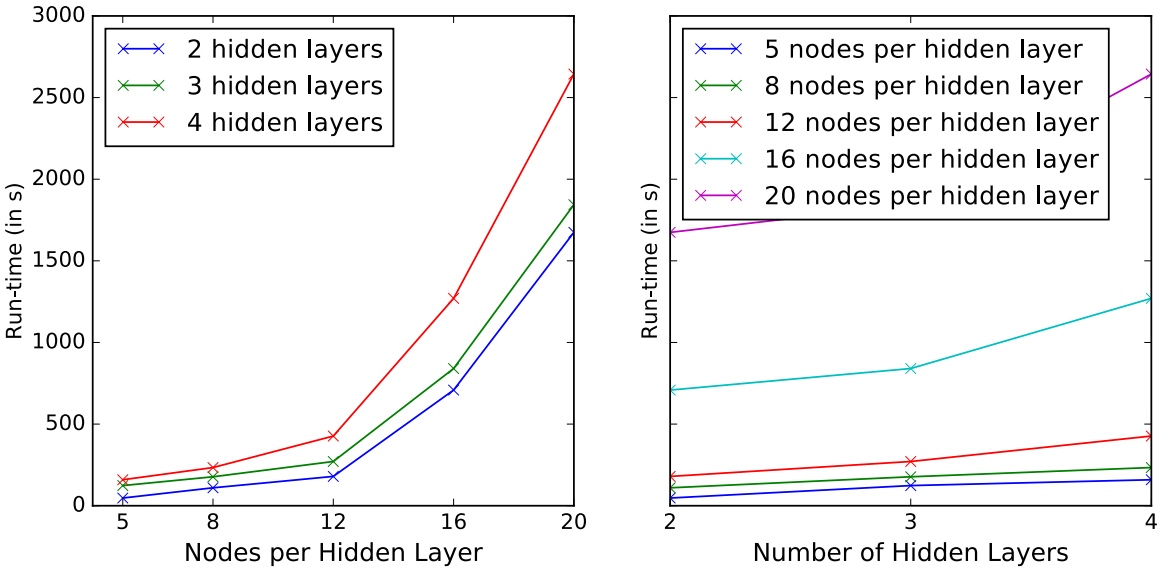
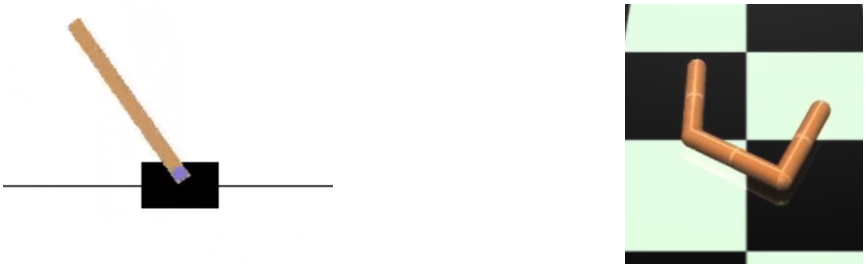


Figure 4.3: Runtimes for computing abstractions. Left: Varying number of nodes in every hidden layer. Right: Varying depth of the trained neural network. The run-time scales exponentially with increasing number of nodes, this is because the worst-case complexity of checking the interpolation condition scales exponentially in the size of the hidden layer. The run-time scales almost linearly with increasing depth.



(a) (b)

Figure 4.4: (a) Cart-pole system: the neural network controller is abstracted into simple control laws, (b) Swimmer-robot: the set of inputs corresponding to high-torque outputs are abstracted into a simple representation.

states (e.g., constant or linear for ReLU nodes), in terms of Hamming distance. This results in data points/counterexamples that have similar nonlinear activation patterns being grouped together.

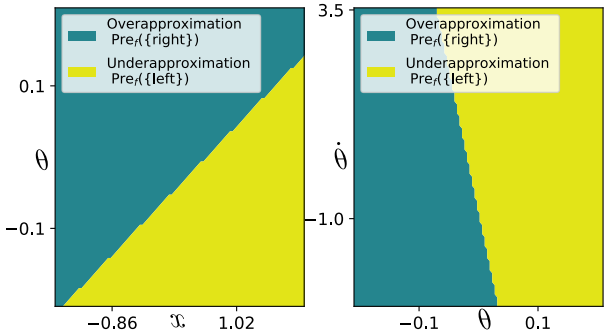


Figure 4.5: Computed abstractions for the neural network cart-pole controller. Left: Varying x, θ with $(\dot{x}, \dot{\theta}) = 0$. Right: Varying $\theta, \dot{\theta}$ with $(x, \dot{x}) = 0$

4.4 Computing Pre-Image Abstractions: Experiments

In this section, we implement and test our approach with neural networks trained on multiple tasks.

4.4.1 2D Toy-Example

We use the simple 2D-example introduced in Section 4.2.3 to study the scalability of the approach. On the same task, we train networks of varying sizes and measure the run times for computing underapproximations of the third quadrant, as classified by the neural network. Figure 4.3 depicts the run times for different network sizes.

Robustness We use the computed underapproximations to verify the robustness of the classifier. Robustness has been extensively studied for classifiers, particularly in image processing [22, 110]. For a given input x and the corresponding output-label k, we say the classifier f is ϵ robust if

$$\forall \bar{x} : \|\bar{x} - x\|_\infty \leq \epsilon, f(\bar{x}) = k.$$

To measure the robustness of the networks trained on this task, we compute both an underapproximation $U^{f,S}$ and an overapproximation $O^{f,S}$ of the pre-image corresponding to third quadrant for a network with 4-hidden layers and 16 nodes per hidden layer. For a set of 50 points from the third quadrant and $\varepsilon = 0.5$ (recall the problem domain is $[-1, 1]$ along each dimension), we check with `z3` for each point if there exists a counterexample

satisfying

$$\|\bar{x} - x\|_\infty \leq \epsilon, \bar{x} \notin U^{f,S}, \text{ and } \|\bar{x} - x\|_\infty \leq \epsilon, \bar{x} \notin O^{f,S}.$$

If a counterexample is found for both conditions, the point is not robust, and if a counterexample is found for the underapproximation but not the overapproximation, the point's robustness is unknown. If no counterexample is found for both conditions, the point is robust. We are able to validate/invalidate the robustness of 49 points and for one point, the result is unknown, and verifying these set of properties using the computed abstractions takes 1.4 s. This shows that the approximations are quite accurate for this task. The computations were performed on a 2.40 GHz quad-core machine with 16 GB of RAM.

4.4.2 Cartpole Control

We consider the classical control problem introduced in [12]. The inputs to the network are observations from a four dimensional state space comprising of the position of the cart (x), the velocity of the cart($\dot{x}$), the angle of the pole (θ) and the angular velocity of the pole ($\dot{\theta}$). We train a neural network with 2-hidden layers for the problem with Deep-Q learning using the environment in [92]. The neural network achieves a reward of perfect score of 200.0, averaged over 100 episodes. The output from the network maps to the discrete actions {`left`, `right`}. For both the output actions, we compute the overapproximations of the pre-image and computing each abstraction takes under 5 minutes. Note that since there are just two output classes, the negation of the overapproximation of one output action results in an underapproximation of the other output action. Both the overapproximations consists of a union of two half-spaces, which implies that the underapproximations are just one single polytope.

On replacing the neural network controller with a controller based on the overapproximation corresponding to {`left`}, the new controller still achieves a perfect score of 200.0. This shows that the abstraction closely matches the exact pre-images for the neural network. Further, these simple abstractions give insight into the internal strategy learned by the neural network. The computed overapproximation for the pre-image of the output set {`left`}

(Pre(`left`)) is

$$(-0.335x - 0.06\dot{x} + 0.918\theta + 0.202\dot{\theta} \leq -0.665)$$
$$\vee(-0.110x + 0.156\dot{x} + 0.950\theta + 0.245\dot{\theta} \leq -0.015).$$

We see that a negative x (the cart is to the left of the workspace), causes the cart to apply a force to the right. A negative θ causes the controller to make the cart move left. (See Figure 4.2.3). This matches the expected intuitive behavior, and is an interpretable abstraction.

Further, we can use the approximations to verify properties about the neural network. For example, consider the property that $\forall x_t, \dot{x}_t, \theta_t, \dot{\theta}_t$ such that $\|x_t\| \leq 0.1, \|\dot{x}_t\| \leq 0.1, \|\theta_t\| \leq 0.1, \|\dot{\theta}_t\| \leq 0.1$, the condition $\|\dot{\theta}_{t+1}\| \leq 0.5$ holds. First, we can show that for all points that satisfy the overapproximation of Pre(`left`) and the action `left`, the property holds with the cart-pole dynamics. Next, we can repeat a similar procedure with Pre(`right`), to fully verify the neural network controller. We verify this property with `dReal`[55], as it can reason over the sin and cos functions that occur in the dynamics. This computation takes 0.124 s. However, on checking for the condition (with the underapproximations) $\|\dot{\theta}_{t+1}\| \leq 0.3$, the solver finds a counterexample with $(x_t, \dot{x}_t, \theta_t, \dot{\theta}_t) = (-0.09, -0.09, 0.0, 0.097)$ as the initial condition with $\dot{\theta}_{t+1} = -0.31$ in 0.037 s.

4.4.3 Swimmer

For this task, we construct a compact abstraction that allows for run-time monitoring. The setting for the problem is to determine if a noisy observation by a monitor could possibly lead to unsafe behavior. The controller we consider is a neural network with 2 hidden layers trained with proximal policy optimization [102] on the `Swimmer` environment [93]. The task is to control a 3-link robot in a viscous fluid to make it swim forward as fast as possible. The network (f) maps from an 8-dimensional state space ($x \in \mathbb{R}^8$) to a 2-dimensional space (τ_1, τ_2) corresponding to the joint actuation torques.

For this task, suppose that in the domain $x \in [-2, 2]^8$, the observations made by a run-time monitor are noisy such that $\|x - x_{\text{true}}\|_\infty \leq 0.1$, where x is the observed state and x_{true} is the true state. The controller has access to x_{true}, while the monitor only has the noisy

reading x. We want to construct a monitor that during the operation of the robot flags an input x as unsafe if x is a noisy observation and it is possible that the true state x_{true} can cause a large torque. Formally, x is unsafe if it satisfies:

$$x \in [-2, 2]^8 \wedge \exists \bar{x}. \|\bar{x} - x\|_\infty \leq 0.1$$

$$\wedge f(\bar{x}) = (\tau_1, \tau_2) \wedge |\tau_1| + |\tau_2| \geq 1.0.$$

Since the monitoring is at run-time, the flagging has to be near instantaneous and we would like to avoid reasoning over the entire network. To allow for this, we compute an overapproximating abstraction $\varphi(\bar{x})$ for the set of inputs to the network in the domain $[-2.1, 2.1]^8$ such that the network outputs $|\tau_1| + |\tau_2| \geq 1.0$. The monitor can be set up using $\varphi(\bar{x})$ as follows:

$$x \in [-2, 2]^8 \wedge \exists \bar{x}. \|\bar{x} - x\|_\infty \leq 0.1 \wedge \varphi(\bar{x}).$$

Algorithm 7 computes a $\varphi(\bar{x})$ that has a simple structure such that, for 50 inputs (sampled from observations seen during training) such that $x \in [-2, 2]^8$, the average time per input for checking the condition above with `z3` is 0.14 seconds. This time can further be reduced by parallelizing the check across polytopes.

4.5 Temporal Specifications for Learning Tasks

In the previous section, we developed a tool for analyzing pre-trained networks. Next, we look at scenarios where we know a priori the property we require our neural network to satisfy. In this direction, verified training [56, 59] has shown to be effective for simple input-output robustness properties for neural networks. Here, we extend verified training for neural networks to a richer class of specifications. To illustrate our approach, we consider three temporal properties expressed in bounded-time STL that we want our networks to satisfy. Note that bounded-time STL specifications can be unrolled into specifications in first-order logic without the modal operators.

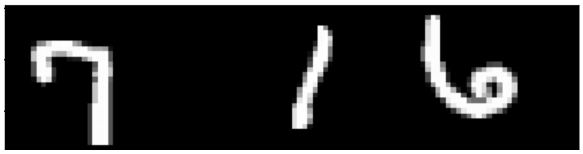


Figure 4.6: MMNIST Image

4.5.1 Bounding Caption Length for Image Captioning

Multi-MNIST images consist of non-overlapping MNIST digits on a canvas of fixed size (Figure 4.6). The number of digits in each image varies between 1 and 3. The task is to label the sequence of digits in the image, followed by an end of sequence token. Prior work on this task [119] has shown image-to-sequence models to be vulnerable to generating sequences longer than the true number of digits in the image, under small adversarially chosen perturbations. Here, we consider the task of training a DNN that does not output sequences longer than the desired length, while achieving similar nominal task performance.

Let $y := f(x)$ be the sequence of logits output by the RNN model when given input image x. For an image x, the termination specification is formalized as follows:

$$\forall \Delta x \in \{s : \|s\|_\infty \leq \epsilon\}.(f(x + \Delta x), 0) \models \varphi_x, \tag{4.8}$$

where $\varphi_x(y) := \Diamond_{[0,t_x^*]} \bigwedge_{i \neq e} (y[t]_e - y[t]_i) \geq 0$, t_x^* is the true number of digits in the image x, e is the label corresponding to the end of sequence token, $\epsilon > 0$ is the perturbation bound. Informally, this specification enforces that the end of sequence token is output no later than after the true number of digits have been output by the RNN, for all inputs within ϵ distance from a true-image.

4.5.2 Verifying That a Robot Never Runs Out of Charge

To demonstrate our approach in the RL setting, we consider a task with a vacuum cleaning robot. We summarize this task here (See Appendix B.1 for more details). The agent (robot) operates in a continuous domain with its location in $(x, y) \in [0, 25]^2$ (Figure 4.7). The room is divided into discrete cells, and the agent gets a reward for visiting any "dirty" cell

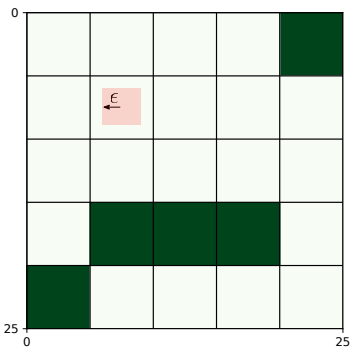


Figure 4.7: Domain for the robot. Recharge cells in green.

which has not been visited in the previous T_{dirt} time-steps. The agent must visit one of the recharge cells every T_{recharge} time-steps, or the episode is terminated with no further reward. The policy maps observations (of the agent location and a map of the room) to continuous velocity controls. We use f_θ to denote the result of applying the policy, parameterized by θ, followed by the environment update.

For this agent, we want to verify the specification: $\forall z \in S_\epsilon.(f_\theta(z), 0) \models \square_{[0,T]} \diamondsuit_{[0,T_{\text{recharge}}]}\varphi_{\text{recharge}}$, where $\varphi_{\text{recharge}}$ corresponds to the agent being in one of the recharge cells. This specification ensures that, for a set of feasible starting positions S_ϵ, for every time-step t in $[0, T]$, the agent recharges itself at least once within T_{recharge} time-steps. *Initial States of the RL Agent* : S_ϵ corresponds to the states (x, y) within a l_∞ distance of ϵ from the center of each of the cells. We formally define this set below.

For a cell i (for Figure 3, $i \in \{1, 2 \ldots 25\}$) with center x_{c_i}, y_{c_i}, the ϵ-ball $S_{i,\epsilon}$ corresponds to the set of positions (x_a, y_a) for the agent such that $\|(x_a - x_{c_i}, y_a - y_{c_i})\|_\infty \leq \epsilon$. Formally,

$$S_{i,\epsilon} := \{(x_a, y_a) : \|(x_a - x_{c_i}, y_a - y_{c_i})\|_\infty \leq \epsilon\}.$$

We can then define $S = \bigcup_i S_{i,\epsilon}$ as the set of feasible initial states of the agent for which we wish to verify the property. Table 4.3 reports the fraction of cells i for which we are able to verify that the agent recharges on starting from $S_{i,\epsilon}$.

4.5.3 Verifying Generated Outputs from a Language Model

A common failure mode for language models is their tendency to fall into degenerate loops, often repeating a stop-word [119]. To illustrate the applicability of STL specifications in this setting, we show how to formalize the property that a GRU language model does not repeat words consecutively. We call this specification bigram non-repetition. More concretely, the desired specification is that the output sequence does not contain bigram repetition amongst the 100 most frequent tokens in the training corpus vocabulary. We want to verify this property over a large set of possible conditioning inputs for the generative model. Concretely, we define an input set S of roughly 25 million prefixes generated from a syntactic template. The prefixes are generated using the following syntax: `<pronoun>, <person>, <action-verb>, <connector>, <person>, <pronoun>, <action-verb>` , where:

`<pronoun>` = {*'my', 'your', 'his', 'her', 'our', 'their'*}

`<person>` = {*'sister', 'brother', 'father', 'mother', 'son', 'daughter', 'king', 'queen', 'knight', 'noble', 'lord', 'duke', 'duchess', 'cousin', 'palace', 'widow', 'nurse', 'marshal', 'archbishop', 'mayor', 'maid'*}

`<action-verb>` = {*['changed', 'despised', 'loved', 'married', 'accused', 'anointed', 'danced', 'rejoiced', 'killed', 'came', 'left', 'prayed', 'stood', 'read', 'consorted', 'denied', 'condemned', 'ruled', 'proved', 'parted' 'resolved', 'committed', 'raised', 'urged', 'painted', 'provoked', 'lived', 'charged', 'yielded', 'accursed', 'assured'],* }

`<connector>` = {*'but', 'while', 'yet', 'and', 'because']*}.

The space of combinations holds 25779600 possibilities to condition the language model generation upon. An example prefix is :'*Our lord yielded and their king left*'. These prefixes are input to the language model, and then we evaluate the specification on the model output.

Now, consider a prefix x and the sequence of logits y output by the recurrent GRU network f (i.e., $y = f(x)$), with $y(t)_k$ referring to the logit corresponding to the k^{th} most frequent token in the vocabulary at time t. A compact formal specification φ_{bigram} ruling

out bigram repetition is

$$\varphi_{\text{bigram}} := \square_{[0,T_{\text{sample}}]} \bigwedge_{i=1,2,\ldots,100} \left(\left(\bigwedge_{j \neq i} y(t)_i \geq y(t)_j \right) \rightarrow \Diamond_{[0,1]} \neg \left(\bigwedge_{j \neq i} y(t)_i \geq y(t)_j \right) \right), \tag{4.9}$$

where T_{sample} denotes the length of the generated sample, in our case 10. The RNN f is required to satisfy the specification $\forall x \in S.(f(x), 0) \models \varphi_{\text{bigram}}$.

4.6 Verified Training of DNNs for STL Specifications

We consider the problem of learning a trace-valued function f_θ to verifiably satisfy a specification of the form $\forall x \in S.\, (f_\theta(x), 0) \models \varphi$, where input x ranges over set S, and $f_\theta(x)$ is the trace generated by f_θ when evaluated on x, θ represents the trainable parameters, and φ is an STL specification. We drop θ for brevity, and simply denote $f_\theta(x)$ as $f(x)$. Formally, our problem statement is:

Given a set of inputs S, train the parameters θ of f_θ so that $\forall x \in S.\, (f_\theta(x), 0) \models \varphi$, where φ is a bounded-time STL specification.

4.6.1 Optimization Formulation of STL Verification

For an STL specification φ, its quantitative semantics can be used to construct a function $\rho(\varphi, f(x), t)$ whose scalar valued output is such that $\rho(\varphi, f(x), t) \geq 0 \iff (f(x), t) \models \varphi$ [41]. In terms of the quantitative semantics, the verification problem is equivalent to showing that $\forall x \in S.\, \rho(\varphi, f(x), 0) \geq 0$. This verification task can be written as the optimization problem of finding the sequence of inputs x such that the sequence of outputs $f(x)$ result in the strongest violation of the specification with regard to the quantitative semantics

$$\min_x \rho(\varphi, f(x), 0) \text{ subject to } x \in S. \tag{4.10}$$

If the solution to equation (4.10) is negative, then there exists an input leading to the violation of φ.

4.6.2 Bound Propagation

The optimization problem in equation (4.10) itself is often intractable; even in the case when the specification is limited to robustness against perturbations in a classification task, it is NP-hard [74]. There are tractable approaches to bounding the problem in equation (4.10) [44, 97], but the bounds are often too loose to provide meaningful guarantees. To obtain a tighter bound tractably, interval bound propagation – which by itself provides loose bounds, but is efficient to compute (2x computational cost) – can be leveraged for verified training to give meaningful bounds on robustness under l_∞ perturbations [59, 90]. Our general approach for doing bound propagation on the function f is to use standard interval arithmetic. While this is straightforward when f is a feedforward DNN [59], here we extend bound propagation to a richer set of (temporal) specifications and architectures. First, we highlight the novel aspects of bound propagation required for (a) auto-regressive RNNs/GRUs, (b) STL specifications.

Bound propagation through GRUs Computing bounds across GRU cells involves propagating bounds through a multiplication operation (as a part of gating mechanisms), which can be handled by a straightforward application of interval arithmetic [66]. Suppose the neural network takes as input x and produces a sequence of outputs y_τ for $\tau = 0, ..., K$ so the overall output is $(y_0, y_1, \ldots, y_K)$. We assume that we are given bounds on the input x:

$$l_0 \leq x \leq u_0.$$

Our goal is to obtain bounds on y_τ given bounds on x for each τ. Each output is produced conditioned on the preceding outputs: y_τ depends on $y_0, \ldots, y_{\tau-1}$.

We proceed recursively, assuming that we have already computed bounds on $y_0, \ldots, y_{\tau-1}$. We stack the set of inputs to the computation as $(x, y_0, \ldots, y_{\tau-1}) \in [l_{\tau;0}, u_{\tau;0}]$. We study the computation graph mapping these inputs to the output y_τ. At each node in this computation graph, we perform a computation of the form

$$z_{\tau,i} = w_i^T h\left(z_{\tau;-i}\right) + \tilde{w}_i^T \tilde{h}\left(z_{\tau;-i}\right)^T z_{\tau;-i} + b_i,$$

where $h, \tilde{h}$ are element-wise nonlinear operations (sigmoid, tanh, ReLU, etc.) and $z_{\tau;-i}$ denotes the elements of computational graph that are ancestors of the node i. The second term represents multiplicative interactions (gating interactions) common in recurrent networks like LSTMs (long short-term memory) [68] and GRUs. Suppose we have already computed lower and upper bounds $l_{\tau;-i}, u_{\tau;-i}$ on the preceding elements. Then, we have

$$
\begin{aligned}
z_{\tau,i} &\geq \max\left(w_i, 0\right)^T h\left(l_{\tau;-i}\right) + \min\left(w_i, 0\right)^T h\left(u_{\tau;-i}\right) \\
&\quad + 1^T \min\begin{pmatrix} \tilde{w}_i \odot \tilde{h}\left(l_{\tau;i}\right) \odot \left(l_{\tau;i}\right), & \tilde{w}_i \odot \tilde{h}\left(u_{\tau;i}\right) \odot \left(l_{\tau;i}\right), \\ \tilde{w}_i \odot \tilde{h}\left(l_{\tau;i}\right) \odot \left(u_{\tau;i}\right), & \tilde{w}_i \odot \tilde{h}\left(u_{\tau;i}\right) \odot \left(u_{\tau;i}\right) \end{pmatrix}, \\
z_{\tau,i} &\leq \max\left(w_i, 0\right)^T h\left(u_{\tau;-i}\right) + \min\left(w_i, 0\right)^T h\left(l_{\tau;-i}\right) \\
&\quad + 1^T \max\begin{pmatrix} \tilde{w}_i \odot \tilde{h}\left(l_{\tau;i}\right) \odot \left(l_{\tau;i}\right), & \tilde{w}_i \odot \tilde{h}\left(u_{\tau;i}\right) \odot \left(l_{\tau;i}\right), \\ \tilde{w}_i \odot \tilde{h}\left(l_{\tau;i}\right) \odot \left(u_{\tau;i}\right), & \tilde{w}_i \odot \tilde{h}\left(u_{\tau;i}\right) \odot \left(u_{\tau;i}\right) \end{pmatrix}.
\end{aligned}
$$

Setting $l_{\tau,i}$ to the lower bound above and $u_{\tau,i}$ to the upper bound, we have computed bounds on $z_{\tau,i}$. Thus, we can recursively compute bounds until we obtain bounds $l_\tau \leq y_\tau \leq u_\tau$, which can then be used to compute bounds on $y_{\tau+1}$. Proceeding recursively, we obtain lower and upper bounds on $(y_0, y_1, \ldots, y_K)$.

Bound propagation through auto-regressive RNNs For language modeling and image captioning, we use GRU decoders with greedy decoding. Greedy decoding involves a composition of the `one-hot` and the `argmax` operations. Both of these operations are non-differentiable. To overcome this and compute differentiable bounds (during training), we approximate this composition with a `softmax` operator (with a low temperature T). In the limit, as $T \to 0$, the softmax operator converges to the composition `one-hot(argmax(`$\cdot$`))` For propagating bounds through the softmax operator, we leverage that the bounds are monotonic in each of the individual inputs. Formally, given a lower ($\underline{p}$) and upper ($\overline{p}$) bound on the input p to a softmax layer (i.e., $\underline{p} \leq p \leq \overline{p}$), the lower $\underline{w}$ and upper bound($\overline{w}$) on the

output can be computed as:

$$\overline{s} = \sum_{i=1}^{N} \exp \overline{p}_i, \quad \underline{s} = \sum_{i=1}^{N} \exp \underline{p}_i, \quad \Delta_i = \exp \overline{p}_i - \exp \underline{p}_i, \quad \overline{w}_i = \frac{\exp \overline{p}_i}{\underline{s} + \Delta_i} \quad \underline{w}_i = \frac{\exp \underline{p}_i}{\overline{s} - \Delta_i},$$

where p_i is the i^{th} coordinate of p and $p \in \mathbb{R}^N$. During evaluation, the `one-hot(argmax(.))` function is used as is. Given bounds on each coordinate of p (i.e., $\underline{p} \leq p \leq \overline{p}$) and $s = \texttt{one-hot}(\texttt{argmax}(p))$, bounds on coordinate s_i can be computed as:

$$\underline{s}_i(x) = \begin{cases} 1 & \underline{p_i} \geq \overline{p}_j . \forall j \\ 0 & \text{otherwise.} \end{cases}$$

$$\overline{s}_i(x) = \begin{cases} 0 & \exists j \neq i \text{ such that, } \underline{p}_j > \overline{p}_i \\ 1 & \text{otherwise.} \end{cases} \tag{4.11}$$

Bounds for discrete inputs Tasks with discrete inputs, such as language generation tasks, encode a prefix sentence as conditioning before decoding a follow-up sequence of words. Consider prefixes of the form $x = x_0, x_1, \ldots$ such that $x_i \in S_i$, where S_i is a finite set of tokens that can appear at position i in the input sequence. We can propagate perturbations in the prefix by first projecting the tokens S_i through the embedding layer E , and then considering the maximum and the minimum value along each embedding dimension to bound the output from E. Formally,

$$\underline{E}_j(x_i) = \min_{x_i \in S_i} E_j(x_i) \leq E_j(x_i) \leq \max_{x_i \in S_i} E_j(x_i) \leq \overline{E}_j(x_i). \tag{4.12}$$

[70, 71] also consider bound propagation for word substitutions.

Bound propagation through the specification First, we extend the quantitative semantics for STL specifications [41] to allow us to reason over sets of inputs. For a STL specification φ in negation normal form (NNF) (See Appendix B.2 for details on the quantitative semantics and conversion to NNF), we first define a lower bound for the quantitative semantics of φ over the set S, which we denote by $\omega_{S,f}(\varphi, 0)$. We define this bound assuming

we have lower bounds on all the atoms occurring in φ. Specifically, let $\Omega_{S,f}(q,t)$ be a lower bound on $q(f(x)_t)$ over all inputs $x \in S$; in other words, at each time t we have $\forall x \in S.\, \Omega_{S,f}(q,t) \leq q(f(x)_t)$. Now, we define the lower bound on a specification φ inductively as:

- $\omega_{S,f}(\texttt{true}, t) = +\infty$,
- $\omega_{S,f}(\neg\texttt{true}, t) = -\infty$,
- $\omega_{S,f}(q(s) \geq 0, t) = \Omega_{S,f}(q,t)$,
- $\omega_{S,f}(\varphi_1 \wedge \varphi_2, t) = \min(\omega_{S,f}(\varphi_1, t), \omega_{S,f}(\varphi_2, t))$,
- $\omega_{S,f}(\varphi_1 \vee \varphi_2, t) = \max(\omega_{S,f}(\varphi_1, t), \omega_{S,f}(\varphi_2, t))$,
- $\omega_{S,f}(\varphi_1 \mathcal{U} \varphi_2, t) = \max\limits_{t' \in t+I} \min\left(\omega_{S,f}(\varphi_2, t'), \min\limits_{t'' \in [t,t']} \omega_{S,f}(\varphi_1, t'')\right)$.

Lemma 20. *For any time t, given lower bounds $\Omega_{S,f}(q,t)$ on all the atoms $q(s) \geq 0$ in φ, we have*

$$\forall x \in S.\, \omega_{S,f}(\varphi, t) \leq \rho(\varphi, f(x), t),$$

Proof. We proceed by induction on φ. The base cases and the conjunction case are straightforward, and the atom case follows by assumption. The disjunction case requires us to show: $\omega_{S,f}(\varphi_1 \vee \varphi_2, t) \leq \min\limits_{x \in S}(\max(\rho(\varphi_1, f(x), t), \rho(\varphi_2, f(x), t)))$. Applying the max–min inequality, the right-hand side is at least $\max(\min\limits_{x \in S}(\rho(\varphi_1, f(x), t)), \min\limits_{x \in S}(\rho(\varphi_2, f(x), t))$. Then using the inductive hypotheses, we know this is at least $\max(\omega_{S,f}(\varphi_1, t), \omega_{S,f}(\varphi_2, t))$, and the case follows. The case for the $\mathcal{U}$ operator has a similar proof based on the max–min inequality. □

Corollary 21. If $\omega_{S,f}(\varphi, t) \geq 0$, then $\forall x \in S.(f(x), 0) \models \varphi$.

In order to compute the lower bounds $\Omega_{S,f}(q,t)$ required for Lemma 20, given bounds on the input x, we can first compute bounds on the outputs $f(x)_t$ at each time t. For the atoms $q(s) \geq 0$ appearing in φ, given bounds on the input s we can compute bounds on $q(s)$. These bounds can then be propagated through the specification inductively.

4.6.3 Verified Training for STL Specifications

In this section, we describe how to train a network to satisfy an STL specification φ. The quantitative semantics $\rho(\varphi, \sigma, 0)$ give a measure of the extent to which σ satisfies φ. First, we compute lower bounds on the values of the atoms in φ at each instance of time. Then, by application of Lemma 20, we can compute the lower bound $\omega_{S,f}(\varphi, 0)$ satisfying $\forall x \in S.\, \omega_{S,f}(\varphi, 0) \leq \rho\,(\varphi, f(x), 0)$. Subsequently we optimize the lower bound $\omega_{S,f}(\varphi, 0)$ to be non-negative, thereby guaranteeing that the specification of interest holds: $\forall x \in S.\, \rho\,(\varphi, f(x), 0) \geq 0$.

Let L_{obj} be the loss objective corresponding to the base task, for example, the cross-entropy loss for classification tasks. Training thus requires balancing two objectives: minimizing loss on the base task by optimizing $L_{obj}(f_\theta)$, and ensuring the positivity of ω_{S,f_θ}. We can use gradient descent to directly optimize the joint loss: $L_{obj}(f_\theta) - \lambda \min\{\omega_{S,f_\theta}(\varphi, 0), \tau\}$, where λ is a scalar hyper-parameter, τ is a positive scalar threshold ($\tau \in \mathbb{R}_+$). The clipping avoids having to carefully balance the two losses. The quantitative semantics of an STL specification φ is a non-smooth function of the weights of the neural network, and is difficult to optimize directly with gradient descent. We find in practice that curriculum training, similar to [59], works best for optimizing the specification loss, starting with enforcing the specification over a subset $S' \subset S$, and gradually covering the entire S. Empirically, the curriculum approach means that the task performance (L_{obj}) does not degrade much.

4.7 Verified Training for STL Specifications: Experiments

4.7.1 Sequential Captioning of Multi-MNIST Images

For this task, we perform verified training to enforce the termination specification φ_x (equation (4.10)) on the training data as discussed in Section 4.6.3. Post training, for unseen test set images, we evaluate the quantitative specification loss $\omega_{S_{x,\epsilon},f}(\varphi_x, 0)$. For an image x from the test set, if $\omega_{S_{x,\epsilon},f}(\varphi_x, 0)$ is positive, it is guaranteed that there is no input within an l_∞ radius of ϵ around the current image that can cause the RNN to generate a longer sequence than the number of true digits in the image.

Table 4.1: Comparison of GRU training methods on the MMNIST task. We evaluate against the termination specification on different metrics, and also report nominal accuracy. '–' indicates a trivial verified accuracy of 0% obtained with bound propagation. The entries with verified termination accuracies corresponding to 0.0 are those where we were able to generate adversarial examples (counterexamples) to the specification for every point in the test set. We found that adversarial training is difficult because of the presence of the sigmoid and tanh activation functions commonly used in GRUs. To have a meaningful baseline, we performed adversarial training on an RNN (feedforward cells with ReLU activation). For $\epsilon = 0.1$, attacking the loss from [119] to produce longer sequences performs better, while for the other ϵ values adversarial training with the STL quantitative loss performs better. Adversarial training performs well but is difficult to verify. At larger ϵ, verified training results in both better guarantees (specification conformance), and better nominal accuracies.

Perturbation ϵ	Training	Nominal Accuracy	Verified Termination Accuracy	Adversarial Termination Accuracy
0.1	Verifiable	94.9	**98.3**	**100.0**
	Adversarial	94.1	–	**100.0**
	Nominal	**95.9**	–	33.5
0.2	Verifiable	94.5	**98.7**	**100.0**
	Adversarial	93.3	–	**100.0**
	Nominal	**95.9**	–	20.94
0.3	Verifiable	94.4	**98.7**	**100.0**
	Adversarial	90.0	–	99.7
	Nominal	**95.9**	0.0	0.0
0.5	Verifiable	94.1	**99.0**	**100.0**
	Adversarial	75.6	–	100.0
	Nominal	**95.9**	0.0	0.0

In Tables 4.1 and 4.2, *verified termination accuracy* refers to the fraction of unseen data for which we can verify the absence of counterexamples to the termination property ((4.10)). *Nominal accuracy* refers to the percentage of correctly predicted tokens, including the end of sequence token. Table 4.1 compares verified training with nominal and adversarial training. Verified training outperforms both adversarial and nominal training on both adversarial and verified termination accuracy metrics. The pixel values are scaled to be in the range $[0, 1]$. At perturbations of size $\epsilon = 0.5$, the images can be turned gray; however, the DNN remains robust to such large perturbations by predicting that the image has no more than a single digit at large perturbations, while maintaining nominal accuracy on clean data. This

Table 4.2: We train the RNN with ReLU activations from [119] to be verifiable with $\epsilon = 0.3$, and compare its verifiability with MILP based verification reported in [119] at different perturbation radii. The nominal accuracy for the model trained to be verifiable is 93.9% and model trained in a standard manner is 96.4%. For larger perturbations, the MILP solver times out. '–' indicates that we were unable to certify robustness for any of the points in the test set, for the given perturbation within the time-out window of 30 minutes.

Perturbation Radius ϵ	Training	Verification Method	Verified Termination Accuracy
0.002	Nominal	MILP	83.00
	Verifiable	Bound Prop.	**99.01**
0.02	Nominal	MILP	–
	Verifiable	Bound Prop.	**98.95**
0.3	Nominal	MILP	–
	Verifiable	Bound Prop.	**94.3**

in contrast with robustness against misclassification, where it is not possible to be robust at large perturbations because the specifications for images from different classes conflict. Adversarial accuracy is evaluated with the iterative attack from [119] (10,000 steps).

Run-time Considerations As another baseline, we compare with verified termination accuracies from [119](Table 4.2). In [119], the greedy decoding and the specification are turned into a MILP-query solved with the SCIP solver [57]. Further, we use ReLU RNNs here because GRUs are not amenable to MILP solvers. Verified training allows us to certify specification conformance for much larger perturbations ($\approx$ 2 orders of magnitude larger).

4.7.2 An RL Mobile-Robot Agent

We consider the recharging specification $\varphi_{\text{recharge}}$ over a time horizon of $T = 10$, for an agent starting within a l_∞ distance of ϵ from the center of the any of the cells. To regularize the DNN to be verifiable with regard to $\varphi_{\text{recharge}}$, the specification loss is obtained by rolling out the current policy through time, and propagating bounds through the rolled out policy and the dynamics. This assumes a deterministic dynamics model.

We compare our verifiably trained agent to both a vanilla RL agent, and an agent trained with reward shaping as in [83]. All agents achieve a similar reward, and we do not find

specification violations for roll-outs from 10^6 random (feasible) initial states. To compare verifiability, we discretize a region within a distance of ϵ to each cell-center into 10^2 l_∞ balls, and verify with bound propagation that the agent satisfies $\varphi_{\text{recharge}}$ for each sub-region. Agents trained with verified training are significantly more verifiable than agents trained otherwise, with little degradation in performance (Table 4.3), which is consistent with prior work in classification [121].

Table 4.3: Mean/Variance performance (across 5 agents of each type) across different metrics. For each agent, reward is computed as mean across 100 episodes. ϵ is distance from the center of the grid cells, and for each ϵ we report the fraction of the cells for which we are able to certify that $\varphi_{\text{recharge}}$ holds.

Training	% of cells verified ($\epsilon = 1.0$)	% of cells verified ($\epsilon = 0.1$)	% of cells verified ($\epsilon = 0.01$)	% of cells verified ($\epsilon = 0.001$)	% of cells verified ($\epsilon = 0.0001$)	Reward
Verifiable	100.0/0.0	100.0/0.0	100.0/0.0	100.0/0.0	100.0/0.0	12.71/0.19
Standard	15.8/9.0	64.9/6.8	77.6/5.3	90.3/1.8	99.2/0.0	12.85/0.06
Reward Shaping	39.1/21.9	74.3/8.6	83.2/5.9	92.0/1.9	100.0/0.0	12.76/0.22

4.7.3 Language Generation

Our language model consists of a 2-layer GRU with 64 hidden nodes per layer, trained on the tiny Shakespeare corpus using a word embedding dimension of 32, and vocabulary truncated to the 2500 most frequent training words. We evaluate the model's ability to satisfy φ_{bigram}. We compare both a nominal model trained using log-likelihood, a model that randomly samples prefixes from the input space and penalizes violations to the specification, and

Table 4.4: Language model perplexity, number of failures during an exhaustive enumerative search over the 25M perturbations, and computational cost of verification (number of forward passes).

Training	Perplexity	# Failures	# Verification Cost
Verifiable	228.91	**0**	$\approx$**2**
Sampled	174.89	**0**	2.57×10^7
Nominal	**153.63**	1.79×10^7	2.57×10^7

verified training that covers the full input space. We report test set perplexity and count of violations observed over the 25 M prefixes (Table 4.4).

We find that while standard training achieves the best perplexity results, it also produces numerous specification failures. Sampling prefixes and regularizing them to avoid bigram repetition using $\rho(\varphi_{\text{bigram}}, f(x), 0)$ eliminates failures, but the overall evaluation cost of the exhaustive search is large. Verifiable training with bound propagation, by contrast, comes with a constant computational cost of ≈ 2 forward passes. This is because matrix multiplications form a significant majority of the computational cost during a forward pass, and propagating bounds through a layer of the form $y = \sigma(Wx + b)$, where σ is a monotonic activation function (e.g., ReLU, sigmoid, tanh), can be performed such that it only costs twice as much as a normal forward pass [59].

Run-time Considerations Verification with propagating bounds can be performed in under 0.4 seconds (including propagating bounds through the spec), while exhaustive search over 25 M prefixes for specification violations takes over 50 minutes. Further, as possible word substitutions increase, the cost for exhaustive search grows exponentially while that for bound propagation stays constant.

4.8 Discussion

Computing Pre-image Abstractions In Section 4.2, we introduced an approach to algorithmically abstract pre-trained neural network pre-images into compact representations that allow for interpretation and verification. The approach introduced here opens several possible directions for future contributions. An interesting direction to explore is if the current approach can be coupled with current verification algorithms for neural networks to improve verification itself. Another avenue to explore is if the abstraction procedure introduced in our work can be coupled with training to learn neural networks that satisfy certain desired properties. Alternatively, given an abstraction for a neural network, an interesting open question is if we can tune the abstraction to satisfy certain desired specifications without compromising significantly on performance.

Verified training for Temporal Specifications In Section 4.6, we developed an approach for training neural networks such that their consistency with temporal specifications (over certain given sets of inputs) can be assured in a tractable manner. We empirically demonstrate the approach on a diverse set of specifications from diverse domains (language processing, image captioning and reinforcement learning), finding that the guarantees we are able to provide are significantly stronger than what can be provided with current tools that analyze neural network behavior based on exhaustive search. Further extensions to the approach would consider training neural networks to be verifiable with regard to probabilistic guarantees on their behavior. This is interesting in scenarios where the behavior of the neural network or the environment is probabilistic (e.g., plants with stochastic dynamics, top-k sampling in language processing).

Chapter 5

Conclusions and Future Work

5.1 Summary

While tools such as SAT solvers have matured considerably over the last couple of decades and have found application at industrial scale [87], more recent tools such as SMT sovlers [37], falsification tools [4], and tools for symbolic model checking [26] still suffer from key-bottlenecks. This thesis presents contributions towards allievating some of the key-challenges limiting the widespread adoption of such tools.

The first contribution is aimed at developing parallelized algorithms for synthesis from linear temporal logic (LTL). This allows one to exploit hardware that allows for parallel computations for faster synthesis. This is accomplished through a compositional approach: first, the primary synthesis problem is decomposed into subproblems, and then the solutions to the subproblems are pieced together to synthesize a strategy that is winning for the primary specification. We identify a special case where the decomposition is straightforward, and a more general case where we use the notion of equicontrollable classes to abstract the problem into a simpler, hierarchical structure.

The second contribution is directed at leveraging learning algorithms for assisting verification. Here, we first introduce an approach for learning abstractions that allow us to verify embedded systems with lookup tables, where the complex lookup tables result in intractable verification problem. We overcome this by learning simple (provable) overapproximations of the lookup tables that render the problem tractable. Building on this, we propose an approach for abstracting instances of nonlinear constraint solving to replace

harder constraints with simpler ones, thereby speeding up the verification process. Besides scalability, another challenging for verification tools is the lack of access to accurate models describing the system's behavior. To allow for the debugging of such systems, which are difficult to model, we propose an approach where learned abstractions that can explain the behavior of such systems with a high probability can be leveraged to ensure the reliablity of controllers interacting with complex systems.

The third direction of contribution is towards developing tools that enable the reliable design and analysis of systems with learned components. Here, we introduce an approach based on Craig's interpolants [30] that allows us to automatically abstract pre-image sets for pre-trained neural networks into simple symbolic formulas that are provable (under) overapproximations for the original pre-image sets. The simpler symbolic abstractions can then be used for interpreting the behavior of the network, and for further analysis with solvers such as [55]. We also introduce tools that allow us to train neural networks such that their consistency with desired temporal specifications can be verified with tractable approaches.

5.2 Future Work

There are many directions and open challenges for future research in enabling the design and verification of reliable autonomous systems at scale. Below we identify some such avenues.

Learning specifications from demonstrations and simulations A key challenge to the adoption of formal methods is the difficulty often associated with crafting the correct specification [61]. In the context of synthesis from assume-guarantee LTL specifications, an interesting direction is learning assumptions about the environment from simulations, and learning guarantees about the system from successful demonstrations. Another promising avenue is controller synthesis from input-output examples. A simple instantiation of this would be one where the end-user feeds the synthesis algorithm a set of initial states, and the corresponding set of desired final states for the system. This is a paradigm that has found

considerable success in the area of program synthesis [62, 82].

Controller Synthesis with probabilistic guarantees In this thesis, we discussed an approach that allows us to provide probabilistic guarantees for a given controller based on simulation data for the environment. In the event the controller is not sufficiently robust or is faulty (does not realize the desired specification), the approach terminates after a set number of iterations by returning a set of counterexamples that exhibit behavior violating the desired specifications. Extending this work, a relevant direction is automatically repairing/synthesizing robust controllers based on the counterexamples. This would allow us to synthesize controllers that are provably probabilistically correct based on simulation data.

Synthesis with human guidance Another interesting paradigm that has received little attention is that of controller synthesis with interactive supervision. Interactive theorem provers, software tools that assist in the development of formal proofs by leveraging human-interaction, have found considerable success [64]. Similar efforts for controller synthesis, where a human could interactively guide the synthesis process are unexplored. A related effort in the program synthesis community is that of synthesizing programs from sketches [106]. *Sketches* are program-skeletons that are human-generated, and a search procedure fills in the *holes* in the sketches to generate a complete program that satisfies the desired specification. The sketches restrict the search space rendering the synthesis problem more tractable. Controller synthesis could benefit from similar approaches where strategies are synthesized to satisfy logic specifications from preliminary solution sketches.

Bibliography

[1] US National Transportation Safety Board. Preliminary report highway: Hwy18mh010. https://www.ntsb.gov/investigations/AccidentReports/Reports/HWY18MH010-prelim.pdf.

[2] A. Albarghouthi and K. L. McMillan. Beautiful interpolants. In N. Sharygina and H. Veith, editors, *Computer Aided Verification*, pages 313–329, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-39799-8.

[3] R. Alur, S. Moarref, and U. Topcu. Compositional synthesis with parametric reactive controllers. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control, HSCC 2016, Vienna, Austria, April 12-14, 2016*, pages 215–224, 2016. doi: 10.1145/2883817.2883842. URL http://doi.acm.org/10.1145/2883817.2883842.

[4] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 254–257. Springer, 2011.

[5] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In P. A. Abdulla and K. R. M. Leino, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 254–257, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-19835-9.

[6] N. Aréchiga, S. Dathathri, S. Vernekar, N. Kathare, S. Gao, and S. Shiraishi. Osiris:

A tool for abstraction and verification of control software with lookup tables. In *Proceedings of the 1st International Workshop on Safe Control of Connected and Autonomous Vehicles*, SCAV'17, pages 11–18, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349765. doi: 10.1145/3055378.3055384. URL https://doi.org/10.1145/3055378.3055384.

[7] J. Ba and R. Caruana. Do deep nets really need to be deep? In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2654–2662. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5484-do-deep-nets-really-need-to-be-deep.pdf.

[8] F. Bacchus, S. Dalmao, and T. Pitassi. Solving #sat and bayesian inference with backtracking search. *J. Artif. Int. Res.*, 34(1):391–442, Mar. 2009. ISSN 1076-9757. URL http://dl.acm.org/citation.cfm?id=1622716.1622727.

[9] R. Baheti and H. Gill. Cyber-physical systems. volume 12, pages 161–166, 2011.

[10] C. Baier and J.-P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008. ISBN 026202649X, 9780262026499.

[11] A. Balkan, P. Tabuada, J. V. Deshmukh, X. Jin, and J. Kapinski. Underminer: A framework for automatically identifying nonconverging behaviors in black-box system models. *ACM Trans. Embed. Comput. Syst.*, 17(1):20:1–20:28, Dec. 2017. ISSN 1539-9087. doi: 10.1145/3122787. URL http://doi.acm.org/10.1145/3122787.

[12] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, Sept 1983. ISSN 0018-9472. doi: 10.1109/TSMC.1983.6313077.

[13] O. Bastani, Y. Pu, and A. Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *Neural Information Processing Systems, NIPS 2018*, 2018.

[14] D. Bezzina and J. Sayer. Safety pilot model deployment: Test conductor team report, USDOT Report No. DOT HS 812 171, 2015.

[15] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive(1) designs. *Journal of Computer and System Sciences*, 78:911–938, May 2012. doi: 10.1016/j.jcss.2011.08.007.

[16] B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45(9):993–1002, 1996. doi: 10.1109/12.537122.

[17] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. ACM. ISBN 0-89791-497-X. doi: 10.1145/130385.130401. URL http://doi.acm.org/10.1145/130385.130401.

[18] A. Browne, E. Clarke, S. Jha, D. Long, and W. Marrero. An improved algorithm for the evaluation of fixpoint expressions. *Theoretical Computer Science*, 178(1):237 – 255, 1997. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/S0304-3975(96)00228-9. URL http://www.sciencedirect.com/science/article/pii/S0304397596002289.

[19] R. Bunel, I. Turkaslan, P. H. S. Torr, P. Kohli, and M. P. Kumar. In *Neural Information Processing Systems, NIPS 2018*, 2018.

[20] G. C. Calafiore. Random convex programs. *SIAM Journal on Optimization*, 20(6): 3427–3464, 2010.

[21] N. Carlini and D. Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 3–14. ACM, 2017.

[22] N. Carlini and D. A. Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA,*

May 22-26, 2017, pages 39–57, 2017. doi: 10.1109/SP.2017.49. URL https://doi.org/10.1109/SP.2017.49.

[23] Y. Chen, S. Dathathri, T. Phan-Minh, and R. M. Murray. Counter-example Guided Learning of Bounds on Environment Behavior. page arXiv:2001.07233, Jan 2020.

[24] S. Chinchali, S. C. Livingston, U. Topcu, J. W. Burdick, and R. M. Murray. Towards formal synthesis of reactive controllers for dexterous robotic manipulation. In *2012 IEEE International Conference on Robotics and Automation*, pages 5183–5189, May 2012. doi: 10.1109/ICRA.2012.6225257.

[25] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

[26] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, volume 2404 of *LNCS*, Copenhagen, Denmark, July 2002. Springer.

[27] A. Cimatti, A. Griggio, and R. Sebastiani. Efficient interpolant generation in satisfiability modulo theories. In C. R. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 397–412, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-78800-3.

[28] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In E. A. Emerson and A. P. Sistla, editors, *Computer Aided Verification: 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000. Proceedings*, pages 154–169, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN 978-3-540-45047-4. doi: 10.1007/10722167_15. URL http://dx.doi.org/10.1007/10722167_15.

[29] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In E. A. Emerson and A. P. Sistla, editors, *Computer Aided*

Verification, pages 154–169, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN 978-3-540-45047-4.

[30] W. Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *J. Sym. Logic*, 3:269–285, 1957.

[31] A. Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1):5 – 41, 2001. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/S0004-3702(00)00069-2. URL http://www.sciencedirect.com/science/article/pii/S0004370200000692. Tradeoffs under Bounded Resources.

[32] S. Dathathri and R. M. Murray. Decomposing GR(1) games with singleton liveness guarantees for efficient synthesis. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 911–917, 2017. doi: 10.1109/CDC.2017.8263775.

[33] S. Dathathri, N. Arechiga, S. Gao, and R. M. Murray. Learning-based abstractions for nonlinear constraint solving. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 592–599, 2017. doi: 10.24963/ijcai.2017/83. URL https://doi.org/10.24963/ijcai.2017/83.

[34] S. Dathathri, S. Gao, and R. M. Murray. Inverse abstraction of neural networks using symbolic interpolation. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 3437–3444. AAAI Press, 2019. doi: 10.1609/aaai.v33i01.33013437. URL https://doi.org/10.1609/aaai.v33i01.33013437.

[35] S. Dathathri, I. Filippidis, and R. M. Murray. Parallelizing synthesis from temporal logic specifications by identifying equicontrollable states. In N. M. Amato, G. Hager, S. Thomas, and M. Torres-Torriti, editors, *Robotics Research*, pages 827–842, Cham, 2020. Springer International Publishing. ISBN 978-3-030-28619-4.

[36] S. Dathathri, A. Madotto, J. Lan, J. Hung, E. Frank, P. Molino, J. Yosinski, and R. Liu. Plug and play language models: A simple approach to controlled text generation. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1edEyBKDS.

[37] 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*, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3-540-78799-2, 978-3-540-78799-0. URL http://dl.acm.org/citation.cfm?id=1792734.1792766.

[38] J. A. DeCastro, J. Alonso-Mora, V. Raman, D. Rus, and H. Kress-Gazit. Collision-free reactive mission and motion planning for multi-robot systems. In *Proceedings of the International Symposium on Robotics Research (ISRR)*, Sestri Levante, Italy, September 2015.

[39] A. Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In T. Touili, B. Cook, and P. Jackson, editors, *Computer Aided Verification*, pages 167–170, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-14295-6.

[40] A. Donzé. On signal temporal logic. In *International Conference on Runtime Verification*, pages 382–383. Springer, 2013.

[41] A. Donzé and O. Maler. Robust satisfaction of temporal logic over real-valued signals. In *Formal Modeling and Analysis of Timed Systems - 8th International Conference, FORMATS 2010, Klosterneuburg, Austria, September 8-10, 2010. Proceedings*, pages 92–106, 2010. doi: 10.1007/978-3-642-15297-9_9. URL https://doi.org/10.1007/978-3-642-15297-9_9.

[42] A. Donzé and O. Maler. Robust satisfaction of temporal logic over real-valued

signals. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 92–106. Springer, 2010.

[43] A. Donzé, T. Ferrère, and O. Maler. Efficient robust monitoring for STL. In N. Sharygina and H. Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 264–279. Springer, 2013. doi: 10.1007/978-3-642-39799-8_19. URL https://doi.org/10.1007/978-3-642-39799-8_19.

[44] K. Dvijotham, R. Stanforth, S. Gowal, T. Mann, and P. Kohli. A dual approach to scalable verification of deep networks. *arXiv preprint arXiv:1803.06567*, 2018.

[45] J. Ebrahimi, A. Rao, D. Lowd, and D. Dou. HotFlip: White-box adversarial examples for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 31–36, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2006. URL https://www.aclweb.org/anthology/P18-2006.

[46] R. Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017, Pune, India, October 3-6, 2017, Proceedings*, pages 269–286, 2017. doi: 10.1007/978-3-319-68167-2_19. URL https://doi.org/10.1007/978-3-319-68167-2_19.

[47] R. Ehlers and V. Raman. *Slugs: Extensible GR(1) Synthesis*, pages 333–339. Springer International Publishing, Cham, 2016. ISBN 978-3-319-41540-6. doi: 10.1007/978-3-319-41540-6_18. URL http://dx.doi.org/10.1007/978-3-319-41540-6_18.

[48] E. A. Emerson and C. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Proc. Symp. on Logic in Computer Science*, Cambridge, MA, 1986.

[49] C. Fan, B. Qi, S. Mitra, and M. Viswanathan. DRYVR: data-driven verification and compositional reasoning for automotive systems. *CoRR*, abs/1702.06902, 2017. URL http://arxiv.org/abs/1702.06902.

[50] C. Fan, B. Qi, and S. Mitra. Data-driven formal reasoning and their applications in safety analysis of vehicle autonomy features. *IEEE Design Test*, 35(3):31–38, June 2018. ISSN 2168-2356. doi: 10.1109/MDAT.2018.2799804.

[51] I. Filippidis, S. Dathathri, S. C. Livingston, N. Ozay, and R. M. Murray. Control design for hybrid systems with TuLiP: The temporal logic planning toolbox. In *2016 IEEE Conference on Control Applications (CCA)*, pages 1030–1041, Sept 2016. doi: 10.1109/CCA.2016.7587949.

[52] A. L. Friesen and P. Domingos. Recursive decomposition for nonconvex optimization. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 253–259. AAAI Press, 2015. ISBN 978-1-57735-738-4. URL http://dl.acm.org/citation.cfm?id=2832249.2832284.

[53] S. Gao, S. Kong, and E. Clarke. dreal: An smt solver for nonlinear theories of the reals (tool paper). In *Conference on Automated Deduction*, 2013.

[54] S. Gao, S. Kong, and E. M. Clarke. *dReal: An SMT Solver for Nonlinear Theories over the Reals*, pages 208–214. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-38574-2. doi: 10.1007/978-3-642-38574-2_14. URL http://dx.doi.org/10.1007/978-3-642-38574-2_14.

[55] S. Gao, S. Kong, and E. M. Clarke. dReal: An SMT solver for nonlinear theories over the reals. In M. P. Bonacina, editor, *Automated Deduction – CADE-24*, pages 208–214, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-38574-2.

[56] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. Ai 2: Safety and robustness certification of neural networks with abstract interpretation. In *IEEE Symposium on Security and Privacy*, 2018.

[57] A. Gleixner, M. Bastubbe, L. Eifler, T. Gally, G. Gamrath, R. L. Gottwald, G. Hendel, C. Hojny, T. Koch, M. E. Lübbecke, S. J. Maher, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, F. Schlösser, C. Schubert, F. Serrano, Y. Shinano, J. M. Viernickel, M. Walter, F. Wegscheider, J. T. Witt, and J. Witzig. The SCIP Optimization Suite 6.0. Technical report, Optimization Online, July 2018. URL http://www.optimization-online.org/DB_HTML/2018/07/6692.html.

[58] R. Gorcitz, E. Kofman, T. Carle, D. Potop-Butucaru, and R. de Simone. *On the Scalability of Constraint Solving for Static/Off-Line Real-Time Scheduling*, pages 108–123. Springer International Publishing, Cham, 2015. ISBN 978-3-319-22975-1. doi: 10.1007/978-3-319-22975-1_8. URL http://dx.doi.org/10.1007/978-3-319-22975-1_8.

[59] S. Gowal, K. Dvijotham, R. Stanforth, R. Bunel, C. Qin, J. Uesato, R. Arandjelovic, T. A. Mann, and P. Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *CoRR*, abs/1810.12715, 2018. URL http://arxiv.org/abs/1810.12715.

[60] H. P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik. Parallel support vector machines: The cascade svm. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 521–528. MIT Press, Cambridge, MA, 2004. URL http://books.nips.cc/papers/files/nips17/NIPS2004_0190.pdf.

[61] S. Gulwani. Automating string processing in spreadsheets using input-output examples. *SIGPLAN Not.*, 46(1):317–330, Jan. 2011. ISSN 0362-1340. doi: 10.1145/1925844.1926423. URL https://doi.org/10.1145/1925844.1926423.

[62] S. Gulwani, W. R. Harris, and R. Singh. Spreadsheet data manipulation using examples. *Commun. ACM*, 55(8):97–105, Aug. 2012. ISSN 0001-0782. doi: 10.1145/2240236.2240260. URL https://doi.org/10.1145/2240236.2240260.

[63] S. Haesaert, P. M. J. V. den Hof, and A. Abate. Data-driven and model-based

verification: a bayesian identification approach. *CoRR*, abs/1509.03347, 2015. URL http://arxiv.org/abs/1509.03347.

[64] J. Harrison, J. Urban, and F. Wiedijk. *History of Interactive Theorem Proving*, volume 9, pages 135–214. 12 2014. doi: 10.1016/B978-0-444-51624-4.50004-6.

[65] E. Helly. Über mengen konvexer körper mit gemeinschaftlichen punkte. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 32:175–176, 1923. URL http://eudml.org/doc/145659.

[66] T. Hickey, Q. Ju, and M. H. Van Emden. Interval arithmetic: From principles to implementation. *J. ACM*, 48(5):1038–1068, Sept. 2001. ISSN 0004-5411. doi: 10.1145/502102.502106. URL http://doi.acm.org/10.1145/502102.502106.

[67] C. A. R. Hoare. An axiomatic basis for computer programming. In *Communications of the ACM*, 1969.

[68] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8): 1735–1780, Nov. 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

[69] A. B. Hocking, M. A. Aiello, J. C. Knight, and N. Aréchiga. Input space partitioning to enable massively parallel proof. In *NASA Formal Methods Symposium*, 2017.

[70] P.-S. Huang, R. Stanforth, J. Welbl, C. Dyer, D. Yogatama, S. Gowal, K. Dvijotham, and P. Kohli. Achieving Verified Robustness to Symbol Substitutions via Interval Bound Propagation. *arXiv e-prints*, art. arXiv:1909.01492, Sep 2019.

[71] R. Jia, A. Raghunathan, K. Göksel, and P. Liang. Certified Robustness to Adversarial Word Substitutions. *arXiv e-prints*, art. arXiv:1909.00986, Sep 2019.

[72] X. Jin, A. Donzé, J. V. Deshmukh, and S. A. Seshia. Mining requirements from closed-loop control models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(11):1704–1717, Nov 2015. ISSN 0278-0070. doi: 10.1109/TCAD.2015.2421907.

[73] D. Jovanović and L. de Moura. Solving non-linear arithmetic. *ACM Commun. Comput. Algebra*, 46(3/4):104–105, Jan. 2013. ISSN 1932-2240. doi: 10.1145/2429135.2429155. URL http://doi.acm.org/10.1145/2429135.2429155.

[74] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[75] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification*, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.

[76] Y. Kesten, N. Piterman, and A. Pnueli. Bridging the gap between fair simulation and trace inclusion. *Information and Computation*, 200:35–61, 2005. doi: 10.1016/j.ic.2005.01.006.

[77] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

[78] C. Ko, Z. Lyu, T. Weng, L. Daniel, N. Wong, and D. Lin. POPQORN: quantifying robustness of recurrent neural networks. *CoRR*, abs/1905.07387, 2019. URL http://arxiv.org/abs/1905.07387.

[79] A. Kozarev, J. Quindlen, J. How, and U. Topcu. Case studies in data-driven verification of dynamical systems. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, HSCC '16, pages 81–86, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3955-1. doi: 10.1145/2883817.2883846. URL http://doi.acm.org/10.1145/2883817.2883846.

[80] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Where's Waldo? Sensor-Based Temporal Logic Motion Planning. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3116–3121, April 2007. doi: 10.1109/ROBOT.2007.363946.

[81] M. Kuderer, S. Gulati, and W. Burgard. Learning driving styles for autonomous vehicles from demonstration. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2641–2646, May 2015. doi: 10.1109/ICRA.2015.7139555.

[82] V. Le and S. Gulwani. Flashextract: A framework for data extraction by examples. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '14, pages 542–553, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450327848. doi: 10.1145/2594291.2594333. URL https://doi.org/10.1145/2594291.2594333.

[83] X. Li, C. I. Vasile, and C. Belta. Reinforcement learning with temporal logic rewards. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*, pages 3834–3839, 2017. doi: 10.1109/IROS.2017.8206234. URL https://doi.org/10.1109/IROS.2017.8206234.

[84] S. C. Livingston. gr1c: a collection of tools for GR(1) synthesis and related activities. http://scottman.net/2012/gr1c. [Online; accessed 15-March 2016].

[85] S. P. Lloyd. Least squares quantization in pcm. *IEEE Trans. Inf. Theor.*, 28(2): 129–137, Sept. 2006. ISSN 0018-9448. doi: 10.1109/TIT.1982.1056489. URL http://dx.doi.org/10.1109/TIT.1982.1056489.

[86] R. Majumdar. Robots at the edge of the cloud. In *Proceedings of the 22Nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems - Volume 9636*, pages 3–13, New York, NY, USA, 2016. Springer-Verlag New York, Inc. ISBN 978-3-662-49673-2. doi: 10.1007/978-3-662-49674-9_1. URL http://dx.doi.org/10.1007/978-3-662-49674-9_1.

[87] S. Malik and L. Zhang. Boolean satisfiability from theoretical hardness to practical success. *Commun. ACM*, 52(8):76–82, Aug. 2009. ISSN 0001-0782. doi: 10.1145/1536616.1536637. URL https://doi.org/10.1145/1536616.1536637.

[88] S. Maniatopoulos, P. Schillinger, V. Pong, D. C. Conner, and H. Kress-Gazit. Reactive high-level behavior synthesis for an atlas humanoid robot. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4192–4199, May 2016. doi: 10.1109/ICRA.2016.7487613.

[89] Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *(PODC '90) Proceedings of the ninth annual ACM Symposium on Principles of Distributed Computing*, pages 377–408, 1990. doi: 10.1145/93385.93442.

[90] M. Mirman, T. Gehr, and M. Vechev. Differentiable abstract interpretation for provably robust neural networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 3578–3586, 2018.

[91] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[92] OpenAI-CartPole-v0. CartPole-v0. `https://gym.openai.com/envs/CartPole-v0/`, 2018. [Online; accessed 2-Sep-2018].

[93] OpenAI-Swimmer-v2. Swimmer-v2. `https://gym.openai.com/envs/Swimmer-v2/`, 2018. [Online; accessed 2-Sep-2018].

[94] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society. doi: 10.1109/SFCS.1977.32. URL `http://dx.doi.org/10.1109/SFCS.1977.32`.

[95] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '89, pages 179–190, New York, NY, USA, 1989. ACM. ISBN 0-89791-294-2. doi: 10.1145/75277.75293. URL `http://doi.acm.org/10.1145/75277.75293`.

[96] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019.

[97] A. Raghunathan, J. Steinhardt, and P. Liang. Certified defenses against adversarial examples. *arXiv preprint arXiv:1801.09344*, 2018.

[98] A. Raghunathan, J. Steinhardt, and P. Liang. Semidefinite relaxations for certifying robustness to adversarial examples. *CoRR*, abs/1811.01057, 2018. URL http://arxiv.org/abs/1811.01057.

[99] C. Ross and I. Swetlitz. Ibm's watson supercomputer recommended 'unsafe and incorrect' cancer treatments, internal documents show. *Stat News https://www. statnews. com/2018/07/25/ibm-watson-recommended-unsafe-incorrect-treatments*, 2018.

[100] D. Sadigh, S. S. Sastry, S. A. Seshia, and A. D. Dragan. Planning for autonomous cars that leverage effects on human actions. In *Proceedings of Robotics: Science and Systems (RSS)*, June 2016. doi: 10.15607/RSS.2016.XII.029.

[101] K. Schneider. *Verification of Reactive Systems: Formal Methods and Algorithms*. SpringerVerlag, 2004. ISBN 3540002960.

[102] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

[103] R. Sharma, A. V. Nori, and A. Aiken. Interpolants as classifiers. In *Computer Aided Verification*, 2012.

[104] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016. URL http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.

[105] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[106] A. Solar-Lezama. The sketching approach to program synthesis. In Z. Hu, editor, *Programming Languages and Systems*, pages 4–13, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-10672-9.

[107] P. Stanley-Marbell, P. Andrea Francese, and M. Rinard. Encoder logic for reducing serial i/o power in sensors and sensor hubs. In *28th Annual IEEE Symposium on High-Performance Chips (Hot Chips'16)*, August 2016.

[108] K. Strabala, M. K. Lee, A. Dragan, J. Forlizzi, and S. S. Srinivasa. Learning the communication of intent prior to physical collaboration. In *2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication*, pages 968–973, Sep. 2012. doi: 10.1109/ROMAN.2012.6343875.

[109] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[110] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013. URL `http://arxiv.org/abs/1312.6199`.

[111] K. M. Ting and L. Zhu. *Boosting Support Vector Machines Successfully*, pages 509–518. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-02326-2. doi: 10.1007/978-3-642-02326-2_51. URL `http://dx.doi.org/10.1007/978-3-642-02326-2_51`.

[112] V. Tjeng, K. Xiao, and R. Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.

[113] P. Trautman, J. Ma, R. M. Murray, and A. Krause. Robot navigation in dense human crowds: the case for cooperation. In *2013 IEEE International Conference on Robotics and Automation*, pages 2153–2160, May 2013. doi: 10.1109/ICRA.2013.6630866.

[114] J. Uesato, B. O'Donoghue, A. v. d. Oord, and P. Kohli. Adversarial risk and the dangers of evaluating against weak attacks. *arXiv preprint arXiv:1802.05666*, 2018.

[115] US National Highway Traffic Safety Administration. Investigation pe 16-007. https://static.nhtsa.gov/odi/inv/2016/INCLA-PE16007-7876.PDF.

[116] M. Vazquez-Chanlatte, J. V. Deshmukh, X. Jin, and S. A. Seshia. Logical clustering and learning for time-series data. In *International Conference on Computer Aided Verification*, pages 305–325. Springer, 2017.

[117] M. Vazquez-Chanlatte, S. Jha, A. Tiwari, M. K. Ho, and S. Seshia. Learning task specifications from demonstrations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 5368–5378. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7782-learning-task-specifications-from-demonstrations.pdf.

[118] A. Verma, V. Murali, R. Singh, P. Kohli, and S. Chaudhuri. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning, ICML 2018*, 2018.

[119] C. Wang, R. Bunel, K. Dvijotham, P.-S. Huang, E. Grefenstette, and P. Kohli. Knowing when to stop: Evaluation and verification of conformity to output-size specifications. In *CVPR*, 2019.

[120] A. Weiser and S. E. Zarantonello. A note on piecewise linear and multilinear table interpolation in many dimensions. *Mathematics of Computation*, 50(181):189–196, 1988. ISSN 00255718, 10886842. URL http://www.jstor.org/stable/2007922.

[121] E. Wong and Z. Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pages 5283–5292, 2018.

[122] S.-H. Wu, K.-P. Lin, C.-M. Chen, and M.-S. Chen. Asymmetric support vector machines: Low false-positive learning under the user tolerance. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data*

Mining, KDD '08, pages 749–757, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-193-4. doi: 10.1145/1401890.1401980. URL http://doi.acm.org/10.1145/1401890.1401980.

[123] S.-H. Wu, K.-P. Lin, H.-H. Chien, C.-M. Chen, and M.-S. Chen. On generalizable low false-positive learning using asymmetric support vector machines. *IEEE Transactions on Knowledge and Data Engineering*, 25(5):1083–1096, 2013. ISSN 1041-4347. doi: http://doi.ieeecomputersociety.org/10.1109/TKDE.2012.46.

[124] M. Yamaura, N. Aréchiga, and S. Shiraishi. Simulink verification benchmark. https://github.com/Toyota-ITC-SSD/SimulinkVerificationBenchmark, 2017.

[125] R. R. Zakrzewski. Verification of a trained neural network accuracy. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*, volume 3, pages 1657–1662 vol.3, July 2001. doi: 10.1109/IJCNN.2001.938410.

Appendix A

Supplementary Material: Parallelized Synthesis for LTL Specifications

A.1 Proof of Claim 2

Proof. First we show $W_\varphi \subseteq W_{\bar{\varphi}}$. Let f_G^φ be a winning strategy for the condition φ for the set W_φ. We show that f_G^φ is winning for the condition $\bar{\varphi}$ for the set of states W_φ, thereby proving that $W_\varphi \subseteq W_{\bar{\varphi}}$. Consider $\sigma \in \text{Plays}(f_G^\varphi)$ such that $\sigma_0 \in W_\varphi$. By definition,

$$\sigma \models (\theta^a \wedge \theta^e) \wedge \left(\Box\rho^e \wedge \bigwedge_{i=1}^{m} \Box\Diamond\psi_i^{\text{e}} \right) \rightarrow \left(\bigwedge_{j=1}^{n} \Box\Diamond\psi_j^{\text{a}}) \right).$$

If $\sigma \models \bigvee_{i=1}^{m} \Diamond\Box\neg\psi_i^{\text{e}}$ or $\sigma \models \neg(\theta^a \wedge \theta^e)$, then $\sigma \models \bar{\varphi}$ directly. In the other case, $\sigma \models \left(\bigwedge_{j=1}^{n} \Box\Diamond\psi_j^{\text{a}} \right)$ has to hold. For this case, the semantics of the $\Diamond$ operator imply that there exists $k_1, k_2, \ldots, k_n$ that are finite such that $\sigma_{k_i} \models \psi_i^{\text{a}}$ ($\Box\Diamond\psi_i^{\text{a}}$ implies $\Diamond\psi_i^{\text{a}}$ has to hold at every step along a run). For each such finite k_i, $\forall j \in \{1, 2, \ldots, n\} \exists h_{ij} > k_i . \sigma_{h_{ij}} \models \psi_j^{\text{a}}$. This implies $\Diamond(\psi_i^{\text{a}} \wedge \Diamond\psi_{i\oplus 1}^{\text{a}})$ holds for each i. Therefore, $\sigma_0 \in W_{\bar{\varphi}}$ and hence $W_\varphi \subseteq W_{\bar{\varphi}}$.

Now, let us show $W_{\bar{\varphi}} \subseteq W_\varphi$. $W_{\bar{\varphi}}$ is the winning set for $\bar{\varphi}$. By definition of winning sets there exists a winning strategy $f_G^{\bar{\varphi}}$ that is winning against $\bar{\varphi}$ for every element of $W_{\bar{\varphi}}$. Also, $W_{\bar{\varphi}}$ is not an empty set if the system can win for $\bar{\varphi}$ from any state. If $W_{\bar{\varphi}}$ is an empty set, $W_{\bar{\varphi}} \subseteq W_\varphi$ is trivially true.

To prove that $W_{\bar{\varphi}} \subseteq W_\varphi$, we construct a new strategy $\bar{f}$ that is winning against the

condition φ for all states in $W_{\bar{\varphi}}$. This way, we show that every state in $W_{\bar{\varphi}}$ is winning against φ and hence in W_{φ}. Consider some play $\bar{\sigma}$ of $f_G^{\bar{\varphi}}$ such that $\bar{\sigma} \models \bigwedge_{i=1}^{m} \square\diamond\psi_i^{\mathrm{e}}$ and $\bar{\sigma}_0 \in W_{\bar{\varphi}}$. If no such play exists, then for all plays of $f_G^{\bar{\varphi}}$, the condition $\neg \bigwedge_{i=1}^{m} \square\diamond\psi_i^{\mathrm{e}}$ holds and the strategy $f_G^{\bar{\varphi}}$ is winning for φ because all plays of $f_G^{\bar{\varphi}}$ satisfy φ.

Consider the case when such a play exists. $\diamond\psi_1^{\mathrm{a}}$ holds implying that at some finite k, $\bar{\sigma}_{k:} \models \psi_1^{\mathrm{a}} \wedge \diamond\psi_{1\oplus 1}^{\mathrm{a}}$ holds. Denote the smallest k at which $\bar{\sigma}_k \models \psi_1^{\mathrm{a}} \wedge \diamond\psi_{1\oplus 1}^{\mathrm{a}}$ as k_1. By a similar reasoning, we can go on to define $k_1, k_2, \ldots, k_n$. Next, we introduce a variable $\mathcal{Z}_n$ that can take values in $\{1, 2, \ldots, n\}$ and tracks which of the liveness guarantees have been satisfied. $\mathcal{Z}_n$ is initialized to 1. The strategy

$$\bar{f} : (M \times \{1, 2, \ldots, n\}) \times \Sigma \times \mathcal{P}(\mathrm{AP_e}) \rightarrow$$

$$(M \times \{1, 2, \ldots, n\}) \times \mathcal{P}(\mathrm{AP_a})$$

is constructed as

$$\bar{f}((w, \mathcal{Z}_n), s, s' \cap \mathrm{AP_e}) = ((w', \mathcal{Z}_n'), s' \cap \mathrm{AP_a}),$$

where if $s \models \psi_{\mathcal{Z}_n}^{\mathrm{a}}$,

$$(w', s' \cap \mathrm{AP_a}) = f_G^{\bar{\varphi}}(m_{k_{\mathcal{Z}_n}}^{\bar{\sigma}, f_G^{\bar{\varphi}}}, s, s' \cap \mathrm{AP_e}),$$

$$\mathcal{Z}_n' = \mathcal{Z}_n \oplus 1,$$

and if $s \not\models \psi_{\mathcal{Z}_n}^{\mathrm{a}}$,

$$(w', s' \cap \mathrm{AP_a}) = f_G^{\bar{\varphi}}(w, s, s' \cap \mathrm{AP_e}),$$

$$\mathcal{Z}_n' = \mathcal{Z}_n.$$

Showing well-definedness for all relevant inputs For any reachable state-memory pair (s, w) of $f_G^{\bar{\varphi}}$ and any input $x \in \mathcal{P}(\mathrm{AP_e})$, $f_G^{\bar{\varphi}}(w, s, x)$ is defined if $(s, x) \models \rho^{\mathrm{e}}$ (since $f_G^{\bar{\varphi}}$ is winning for $\bar{\varphi}$). For the case when (s, w) is reachable, then $f(w, s, x)$ is also reachable if $sx \models \rho^{\mathrm{e}}$. This implies that when $s \not\models \psi_{\mathcal{Z}_n}^{\mathrm{a}}$ if $ss' \models \rho^{\mathrm{e}}$ and (s, w) is reachable, then (w', s')

with $(w', s' \cap \mathrm{AP_a}) = \bar{f}(w, s, s' \cap \mathrm{AP_e})$ is reachable.

Consider a state s such that $s \models \psi^{\mathrm{a}}_{\mathcal{Z}_n}$, $s = \bar{\sigma}_{k_{\mathcal{Z}_n}}$ because $\bar{\sigma}_{k_{\mathcal{Z}_n}} \models \psi^{\mathrm{a}}_{\mathcal{Z}_n}$ and $[[\psi^{\mathrm{a}}_{\mathcal{Z}_n}]]$ is a singleton. If $s = \bar{\sigma}_{k_{\mathcal{Z}_n}}$, then $f^{\bar{\varphi}}_G(m^{\bar{\sigma}, f^{\bar{\varphi}}_G}_{k_{\mathcal{Z}_n}}, s, x)$ is defined $\forall x \in \mathcal{P}(\mathrm{AP_e}).sx \models \rho^{\mathrm{e}}$. This is because $(s, m^{\bar{\sigma}, f^{\bar{\varphi}}_G}_{k_{\mathcal{Z}_n}})$ is reached during the execution $\bar{\sigma} \in Plays(f^{\bar{\varphi}}_G)$. Therefore, for any $s \models \psi^{\mathrm{a}}_{\mathcal{Z}_n}$, $\bar{f}(m^{\bar{\sigma}, f^{\bar{\varphi}}_G}_{k_{\mathcal{Z}_n}}, s, (.))$ is well-defined for all valid environmental inputs and $(s, m^{\bar{\sigma}, f^{\bar{\varphi}}_G}_{k_{\mathcal{Z}_n}})$ is reachable for $f^{\bar{\varphi}}_G$.

Additionally, we begin execution for the first input at an initial memory value $m^i \in M$. For a valid initial state $s \in W_{\bar{\varphi}}$ and the initial memory value m^i, (s, m^i) is reachable for $f^{\bar{\varphi}}_G$. To summarize, we start at a reachable state-memory pair for $f^{\bar{\varphi}}_G$.

We showed that for any reachable state-memory pair (s, w) of $f^{\bar{\varphi}}_G$, $\bar{f}$ is well-defined for all valid environmental inputs. We also showed that the output for this case is a reachable state-memory pair (for $f^{\bar{\varphi}}_G$) if the environmental input is valid. Additionally, we also start at a reachable state-memory pair. Therefore, for any $\sigma \in \mathrm{Pref}(\bar{f})$, at $(\sigma_{-1}, m^{\sigma, f}_{-1})$, $\bar{f}$ is well-defined for all valid inputs if $\sigma_r \sigma_{r+1} \models \rho^{\mathrm{e}}$ $\forall r < |\sigma| - 1$, $\sigma_0 \in W_{\bar{\varphi}}$, and execution starts with the initial memory value m^i.

Proving properties about the strategy $\bar{f}$ We argued that $\bar{f}$ is input-enabled. This implies that for a state, $\bar{f}$ is well-defined for any valid environmental input when the environment assumption has not been violated in the past while getting to that state. Now all that remains is to show that the plays of $\bar{f}$ satisfy the specification φ.

Consider any $\sigma \in \mathrm{Plays}(\bar{f})$ with $\sigma_0 \in W_{\bar{\varphi}}$. Note that the state sequence $\bar{\sigma}$ used for the construction of the strategy $\bar{f}^{\varphi}_G$ is independent of the sequence of inputs corresponding to σ. Also, note that the strategy $\bar{f}$ and $\mathrm{Plays}(\bar{f})$ have already been defined. Here we only prove properties about elements of the set $\mathrm{Plays}(\bar{f})$, specifically that they satisfy φ. Consider the case when $\sigma \models \bigwedge_{i=1}^{m} \Box\Diamond\psi^{\mathrm{e}}_i$, because for the other case φ holds directly.

Execution begins at a valid initial state m^i and $\sigma_0 \in W_{\bar{\varphi}}$. If $\sigma \not\models \Diamond\psi^{\mathrm{a}}_1$, it implies that execution continued in accordance with $f^{\bar{\varphi}}_G$ without any memory resets (from the definition of $\bar{f}$). This implies that $\sigma \in \mathrm{Plays}(f^{\bar{\varphi}}_G)$, but this leads to a contradiction since $\sigma \models \Diamond\psi^{\mathrm{a}}_1 \wedge \bigwedge_{i=1}^{n} \Diamond(\psi^{\mathrm{a}}_i \rightarrow \Diamond\psi^{\mathrm{a}}_{i\oplus 1})$, implying $\sigma \models \Diamond\psi^{\mathrm{a}}_1$. This is because $\sigma \models \bar{\varphi}$ and we are

looking at the case when $\sigma \models \bigwedge_{i=1}^{m} \square\diamond\psi_i^{\mathrm{e}}$. So, let l_1 be the smallest value at which $\sigma_{l_1} \models \psi_1^{\mathrm{a}}$ holds.

Now consider $\bar{\sigma}_{:k_1}$, the path to $\bar{\sigma}_{k_1}$. Let us look at the sequence $\sigma_{l_1:}$. If $\sigma_{l_1:} \not\models \diamond\psi_2^{\mathrm{a}}$, then the sequence $\bar{\sigma}_{:k_1}\sigma_{l_1:} \in \mathrm{Plays}(f_G^{\bar{\varphi}})$. This is because $|[[\psi_1^{\mathrm{a}}]]|=1$, $\bar{\sigma}_{k_1} = \sigma_{l_1}$. And by construction,

$$\bar{f}((m^{\sigma_{l_1},\bar{f}}, 1), \sigma_{l_1}, \sigma_{l_1+1} \cap \mathrm{AP_e}) = f_G^{\bar{\varphi}}(m_{k_1}^{\bar{\sigma}, f_G^{\bar{\varphi}}}, \bar{\sigma}_{k_1}, \sigma_{l_1+1} \cap \mathrm{AP_e}).$$

Therefore, $\bar{\sigma}_{:k_1}\sigma_{l_1:} \in \mathrm{Plays}(f_G^{\bar{\varphi}})$ and $(\bar{\sigma}_{:k_1}\sigma_{l_1:})_0 \in W_{\bar{\varphi}}$. This implies that

$$\bar{\sigma}_{:k_1}\sigma_{l_1:} \models \psi_1^{\mathrm{a}} \wedge \diamond(\psi_1^{\mathrm{a}} \rightarrow \diamond\psi_2^{\mathrm{a}})$$

from the definition of $\bar{\varphi}$ and $m_{k_1}^{\bar{\sigma}, f_G^{\bar{\varphi}}}$ – leading to a conttradiction to our assumption $\sigma_{l_1:} \not\models \diamond\psi_2^{\mathrm{a}}$. Thus, there exists a finite $l_2 \geq l_1$ at which $\sigma_{l_2} \models \psi_2^{\mathrm{a}}$. The inequality $l_2 \geq l_1$ can be made strict i.e $l_2 > l_1$ by identifying any i, j for which $[[\psi_j^{\mathrm{a}}]] = [[\psi_i^{\mathrm{a}}]]$, and combining them into one progress condition. This means that the same state will not satisfy any two distinct progress conditions, hence $l_2 > l_1$ from the condition $\diamond(\psi_1^{\mathrm{a}} \rightarrow \diamond\psi_2^{\mathrm{a}})$.

Repeating the argument for any i, we get $\exists l_{i\oplus 1} > l_i$ such that l_{i+1} is finite and $\sigma_{l_{i\oplus 1}} \models \psi_{i\oplus 1}^{\mathrm{a}}$ with $\sigma_{l_i} \models \psi_i^{\mathrm{a}}$. This way we showed that there exists a sequence of integers such that $l_1^1 < l_2^1 < \ldots l_n^1 < l_1^2 \ldots < l_j^k \ \forall j \leq n, \forall k$ with $\sigma_{l_k^i} \models \psi_i^{\mathrm{a}}$. Given any $j \leq n$ and $r \in \mathbb{N}$ we can find a k such that $r < (k-1)n$, $\sigma_{l_j^k} \models \psi_j^{\mathrm{a}}$. Therefore, $\sigma_{r:} \models \diamond\psi_j^{\mathrm{a}}$. This holds true for all r and for all $j \leq n$, hence $\sigma \models \bigwedge_{i=1}^{n} \square\diamond\psi_i^{\mathrm{a}}$. Therefore, $\bar{f}$ is winning against φ and $\sigma_0 \in W_{\bar{\varphi}} \rightarrow \sigma_0 \in W_{\varphi}$. Therefore, $W_{\bar{\varphi}} \subseteq W_{\varphi}$. And from before $W_{\varphi} \subseteq W_{\bar{\varphi}}$, hence $W_{\bar{\varphi}} = W_{\varphi}$. □

A.2 Proof of Lemma 3

Proof. We first show that for a game with $\bar{\varphi}$ as the winning condition, we can compute the winning strategy from solving $n+1$ reachability games. Then we use the result from Claim 3. Consider a state s that is winning for $\bar{\varphi}$. Let $\bar{f}_G^{\bar{\varphi}}$ be the winning strategy for $\bar{\varphi}$ from s. Consider

$\sigma \in Plays(\bar{f}_G^{\bar{\varphi}})$, then $\sigma \models \psi_1^{\mathrm{a}} \wedge \bigwedge_{j=1}^{n} \diamond(\psi_j^{\mathrm{a}} \rightarrow \diamond\psi_{j\oplus 1}^{\mathrm{a}})$ or $\sigma \models \bigvee_{i=1}^{m} \diamond\square\neg\psi_i^{\mathrm{e}} \vee \neg(\theta^e \wedge \theta^a)$. If all plays of $\bar{f}_G^{\bar{\varphi}}$ with a valid initial state s satisfy $\bigvee_{i=1}^{m} \diamond\square\neg\psi_i^{\mathrm{e}}$, then $\bar{f}_G^{\bar{\varphi}}$ is winning for $\varphi_0^{\mathrm{reach}}$ as well from s. Therefore, by solving the reachability game with $\varphi_0^{\mathrm{reach}}$ as the winning condition, we can obtain a strategy winning for φ.

Consider the case when $\exists\sigma \in \mathrm{Plays}(\bar{f}_G^{\bar{\varphi}})$ such that $\sigma \models \bigwedge_{i=1}^{m} \square\diamond\psi_i^{\mathrm{e}}$ and $\sigma_0 = s$ where $s \models (\theta^a \wedge \theta^e)$. We observe that for this case, for each $i \in \{1, 2, \ldots, n\}$, the reachability game with the condition $\varphi_i^{\mathrm{reach}}$ is winnable from $[[\psi_{i\oplus -1}^{\mathrm{a}}]]$. To do this, first note

$$\sigma \models \psi_1^{\mathrm{a}} \wedge \bigwedge_{j=1}^{n} \diamond(\psi_j^{\mathrm{a}} \rightarrow \diamond\psi_{j\oplus 1}^{\mathrm{a}})$$

since $\sigma \models \bigwedge_{i=1}^{m} \square\diamond\psi_i^{\mathrm{e}} \wedge (\theta^e \wedge \theta^a)$ and $\sigma \models \bar{\varphi}$. Define r_i to be the smallest instance such that $\sigma_{r_i} \models \psi_i^{\mathrm{a}}$ (we know that such an r_i exists from the arguments in the proof of Claim 3).

Next, we prove that the strategy $\bar{f}_G^{\bar{\varphi}}$ with the initial memory value as $m_{r_i}^{\sigma, \bar{f}_G^{\bar{\varphi}}}$ is winning for game with condition $\varphi_{i\oplus 1}^{\mathrm{reach}}$. To see this, consider any $\underline{\sigma} \in \mathrm{Plays}(\bar{f}_G^{\bar{\varphi}})$ with $\underline{\sigma}_0 = q$ such that $q \models \psi_i^{\mathrm{a}}$ i.e execute according to the strategy starting at the state q and memory $m_{r_i}^{\sigma, \bar{f}_G^{\bar{\varphi}}}$. Note that since $[[\psi_i^{\mathrm{a}}]]$ is a singleton, $q = \sigma_{r_i} = \underline{\sigma}_0$. And, at the state-memory value pair $(q, m_{r_i}^{\sigma, \bar{f}_G^{\bar{\varphi}}})$, $\bar{f}_G^{\bar{\varphi}}$ is well-defined since this state-memory value pair is reachable for $\bar{f}_G^{\bar{\varphi}}$ (recall we selected this state and memory value from a execution in $Plays(\bar{f}_G^{\bar{\varphi}})$ starting from the winning set for $\bar{\varphi}$).

Case 1: $\underline{\sigma} \models \bigwedge_{i=1}^{m} \square\diamond\psi_i^{\mathrm{e}}$

For this case, $\sigma_{:r_i}\underline{\sigma} \in Plays(\bar{f}_G^{\bar{\varphi}})$ since $\forall k < r_i, (m_{k+1}^{\sigma}, \sigma_{k+1} \cap \mathrm{AP_a}) = \bar{f}_G^{\bar{\varphi}}(m_k^{\sigma}\sigma_k, \sigma_{k+1} \cap \mathrm{AP_e})$ and since $\sigma_{r_i} = \underline{\sigma}_0$. We continue execution from $(\sigma_{r_i}, m_{r_i}^{\sigma, \bar{f}_G^{\bar{\varphi}}})$ in accordance with $\bar{f}_G^{\bar{\varphi}}$, so the entire sequence was generated in accordance with this strategy. Using the fact that this strategy is winning from σ_0 for $\bar{\varphi}$, and that the semantics of LTL imply

$$\underline{\sigma} \models \bigwedge_{i=1}^{m} \square\diamond\psi_i^{\mathrm{e}} \rightarrow \sigma_{:r_i}\underline{\sigma} \models \bigwedge_{i=1}^{m} \square\diamond\psi_i^{\mathrm{e}},$$

we arrive at the conclusion that

$$\sigma_{:r_i}\underline{\sigma} \models \psi_1^{\mathrm{a}} \wedge \bigwedge_{j=1}^{n} \Diamond(\psi_j^{\mathrm{a}} \rightarrow \Diamond\psi_{j\oplus 1}^{\mathrm{a}}).$$

Therefore, $\sigma_{:r_i}\underline{\sigma} \models (\psi_i^{\mathrm{a}} \wedge \Diamond\psi_{i\oplus 1}^{\mathrm{a}})$ and r_i was the smallest instance at which ψ_i^{a} holds. It follows that $\underline{\sigma} \models (\psi_i^{\mathrm{a}} \wedge \Diamond\psi_{i\oplus 1}^{\mathrm{a}})$. Therefore, $\underline{\sigma} \models \varphi_{i\oplus 1}^{\mathrm{reach}}$.

Case 2: $\underline{\sigma} \models \neg \bigwedge_{i=1}^{m} \Box\Diamond\psi_i^{\mathrm{e}}$

$\underline{\sigma} \models \neg \bigwedge_{i=1}^{m} \Box\Diamond\psi_i^{\mathrm{e}} \rightarrow \underline{\sigma} \models \varphi_i^{\mathrm{reach}}$.

This implies that all plays of $f_G^{\bar{\varphi}}$ starting with $s \models \psi_i^{\mathrm{a}}$ and initial memory value $m_{r_i}^{\sigma, f_G^{\bar{\varphi}}}$ are winning against $\varphi_i^{\mathrm{reach}}$. Hence, $f_G^{\bar{\varphi}}$ is winning against $\varphi_i^{\mathrm{reach}}$ for the state $s \models \psi_i^{\mathrm{a}}$. For the case with $s \models \theta$, by the definition of $\bar{\varphi}$, the set of states $[[\theta]]$ is winning for $\varphi_n^{\mathrm{reach}}$.

Now, we have shown that for the case when the reachability game $\varphi_0^{\mathrm{reach}}$ is not winnable, if $\bar{\varphi}$ is winnable, we can find a winning strategy for each of the $\varphi_j^{\mathrm{reach}}$ games with their respective initial conditions as described in Section 2.2. Let the winning strategy for each such reachability game be $f_G^{\mathrm{reach}_j} : M^j \times \Sigma \times \mathcal{P}(\mathrm{AP_e}) \rightarrow M^j \times \mathcal{P}(\mathrm{AP_a})$ with m_0^i as the initial memory. Without loss of generality, assume that the for any i, j with $i \neq j$, $M^i \cap M^j = \emptyset$. The earlier segment of the proof was to show the existence of these strategies when $\bar{\varphi}$ is winnable.

Now we show these can be combined to form a winning strategy $f_G^{\bar{\varphi}}$ winning against $\bar{\varphi}$. First, consider the strategy f^{reach_n}. Replace all the memory values corresponding to reachable (w, s) for f^{reach_n} where $s \models \psi_1^{\mathrm{a}}$ with m_0^1. Note that $s \models \psi_1^{\mathrm{a}}$ corresponds to a valid initial state for the game with condition $\varphi_2^{\mathrm{reach}}$.

Let this modified strategy be $\bar{f}^{\mathrm{reach}_n}$. Effectively, we have patched f^{reach_n} with f^{reach_1} so that after reaching a state that satisfies ψ_1^{a} it switches from f^{reach_n} to f^{reach_1}. Call the new resulting strategy $f_{1,2}^*$ where $f_{1,2}^*$ is defined as

$$f_{1,2}^*(w', y) = \begin{cases} \bar{f}^{\mathrm{reach}_n}(w, s, x) & \text{if } w \in M^n, \\ f^{\mathrm{reach}_1}(w, s, x) & \text{if } w \in M^1. \end{cases}$$

Consider σ in Plays($f^*_{1,2}$) such that $\sigma \models (\theta^a \wedge \theta^e) \wedge \bigwedge_{i=1}^{m} \square\diamond\psi_i^{\mathrm{e}}$ (the other case is trivial). Initially, we start at memory m_0^n and a state $\sigma_0 : \sigma_0 \models (\theta^e \wedge \theta^a)$ and continue execution along strategy f^{reach_n} till we reach ψ_1^{a} in a finite number of steps–this is guaranteed by the definition of $\varphi_n^{\mathrm{reach}}$. Subsequently, execution is continued along f^{reach_1} till we reach a state that satisfies ψ_2^{a} in a finite number of steps. This is because if $\sigma \models \bigwedge_{i=1}^{m} \square\diamond\psi_i^{\mathrm{e}}$, then $\sigma \models \diamond\psi_1^{\mathrm{a}} \wedge \diamond(\psi_1^{\mathrm{a}} \rightarrow \diamond\psi_2^{\mathrm{a}})$. Otherwise, from the definition of f_1^{reach} and $\varphi_2^{\mathrm{reach}}$, we end up with a contradiction as before. Similarly, we extend $f^*_{1,2}$ to replace the memory corresponding to the reachable (w, s) for f_1^{reach} where $s \models \psi_2^{\mathrm{a}}$ with m_0^2. Define the resulting strategy $f^*_{1,2,3}$ as:

$$f^*_{1,2,3}(w', y) = \begin{cases} \bar{f}^{\mathrm{reach}_n}(w, s, x) & \text{if } w \in M^n, \\ \bar{f}^{\mathrm{reach}_1}(w, s, x) & \text{if } w \in M^1, \\ f^{\mathrm{reach}_2}(w, s, x) & \text{if } w \in M^2. \end{cases}$$

As before, we can show that the plays of this strategy are winning against $\diamond\psi_1^{\mathrm{reach}} \wedge \diamond(\psi_1^{\mathrm{reach}} \rightarrow \diamond\psi_2^{\mathrm{reach}}) \wedge \diamond(\psi_2^{\mathrm{reach}} \rightarrow \diamond\psi_3^{\mathrm{reach}})$. Continue the procedure to obtain $f^*_{1,2,\ldots,n,1}$. By construction, this stategy is winning against $\bar{\varphi}$.

We argued that for a state $s \models (\theta^a \wedge \theta^e)$ that is winning for $\bar{\varphi}$, either the reachability game with condition φ_0^{reach} is winnable or the reachability games with condition φ_i^{reach} with $i \in \{1, 2, \ldots, n\}$ are winnable. And for both cases, we provided a construction for a strategy winning against $\bar{\varphi}$ from the strategies winning for the reachability games. This implies that for a state, $\bar{\varphi}$ is winnable if and only if the game with φ_0^{reach} is winnable or the games with φ_i^{reach} are winnable. Therefore, from solving the reachability games we can infer if a state is winnable or not and also construct a winning strategy for $\bar{\varphi}$ if it is winnable.

In the proof of Claim 3, we demonstrated an approach to construct a strategy that is winning against φ using the strategy winning against $\bar{\varphi}$. We also showed that the winning states for the conditions φ and $\bar{\varphi}$ are the same. Hence, the GR(1) game can be solved by solving $n + 1$ reachability games separately. □

Appendix B

Supplementary Material: Verifying, Interpreting and Debugging Learned Systems

B.1 RL Agent: Task and Training Details

The agent's observation at each time-step t contains i) its own coordinates (x_t, y_t), ii) for each cell, the time remaining until that cell is dirty, and iii) the (fixed) locations of the recharge cells. The agent learns a policy from these observations to a continuous control action $(a_{x,t}, a_{y,t}) \in \mathbb{R}^2$. The continuous part of the agent's state is updated as:

$$x_{t+1} = x_t + a_{x,t}, \qquad y_{t+1} = y_t + a_{y,t}. \tag{B.1}$$

The policy is represented in the parameters θ, and the result of applying the policy then the environment update is our function f_θ.

Here, we consider verifying an agents trained with Deep-Q learning in an environment with $T_{\text{recharge}} = 3$ and $T_{\text{dirt}} = 4$, i.e, every cell accumulates dust four time-steps after it was cleaned, and the robot needs to recharge itself every 4 time-steps. Additionally, the initial cell where the robot starts can have between $0 - 0.1\%$ uncertainty in the amount of dirt, the charge that can be acquired from the recharge station, and the initial battery (as another element of uncertainty in the initial position).

If the agent leaves the domain, it is clipped back into the problem domain. The agent's

each have 4 discrete actions and are trained with a combination of vanilla Deep-Q learning, Deep-Q learning + reward-shaping, and Deep-Q learning + verifiable training. The actions corresponds to velocities in the 4 cardinal directions, i.e., $\{(0, 5), (5, 0), (-5, 0), (0, -5)\}$. For the agent trained to be verifiable, in addition to loss from Deep-Q learning, the agent is also trained to be verifiable with respect to the temporal specification presented in Section 4.5.2. The verification losses are optimized using the Adam optimizer [77] with learning rate 10^{-3}. The weight of the verification loss anneals linearly between 0 and 1.5 during the first 70K steps. The model is trained to be verifiable starting at the center of the grids, and eventually covering the region around the centers, with $\epsilon = 0.015$ during the same 70k steps. We find that this regularizes the model to be verifiable in regions outside the region in which the model was trained to be verifiable.

B.2 STL Semantics

In addition to the qualitative semantics discussed in the main text, STL formulae have quantitative semantics [42, 43] defined inductively by the function ρ below. For a given trace σ, with σ_t indicating the value of the signal at time t, the quantitative semantics is given by

- $\rho(\texttt{true}, \sigma, t) = +\infty$
- $\rho(q(s) \geq 0, \sigma, t) = q(\sigma_t)$
- $\rho(\neg\varphi, \sigma, t) = -\rho(\varphi, \sigma, t)$
- $\rho(\varphi_1 \wedge \varphi_2, \sigma, t) = \min\left(\rho(\varphi_1, \sigma, t), \rho(\varphi_2, \sigma, t)\right)$
- $\rho(\varphi_1 \,\mathcal{U}_I\, \varphi_2, \sigma, t) = \max\limits_{t' \in t+I} \min\left(\rho(\varphi_2, \sigma, t'), \min\limits_{t'' \in [t,t']} \rho(\varphi_1, \sigma, t'')\right)$

One can obtain the qualitative semantics from the sign of the quantitative semantics. Specifically, $(\sigma, t) \models \varphi \iff \rho(\varphi, \sigma, t) \geq 0$.

We can convert the formulae to their equivalent negation normal form by following the standard procedure until negations are only associated with atoms and Boolean constants.

In particular, we interpret $\rho(\neg \texttt{true}, \sigma, t) = -\infty$ and use the disjunction operator defined as $\rho(\varphi_1 \vee \varphi_2, \sigma, t) = \max(\rho(\varphi_1, \sigma, t), \rho(\varphi_2, \sigma, t))$. The normal form for $\neg(\varphi_1 \mathcal{U}_I \varphi_2)$ is obtained by pushing the negation into the subformulae, and swapping min with max. Finally, we turn $\neg(q(s) \geq 0)$ into $-q(s) > 0$ which we approximate by $-q(s) - \delta \geq 0$ for some small $\delta > 0$.