ARCH-COMP24 Category Report: Artificial Intelligence and Neural Network Control Systems (AINNCS) for Continuous and Hybrid Systems Plants

Diego Manzanas Lopez[1], Matthias Althoff[2], Luis Benet[3], Clemens Blab[4], Marcelo Forets[5], Yuhao Jia[6], Taylor T. Johnson[1], Manuel Kranzl[4], Tobias Ladner[2], Lukas Linauer[4], Philipp Neubauer[4], Sophie A. Neubauer[4], Christian Schilling[7], Huan Zhang[6], and Xiangru Zhong[6]

[1] Vanderbilt University, Nashville, TN
{diego.manzanas.lopez, taylor.johnson}@vanderbilt.edu
[2] Technische Universität München (TUM), Munich, Germany
{althoff, tobias.ladner}@tum.de
[3] Instituto de Ciencias Físicas, Universidad Nacional Autónoma de México (UNAM), México
benet@icf.unam.mx
[4] DatenVorsprung GmbH, Vienna, Austria
team@datenvorsprung.at
[5] Universidad de la República, Montevideo, Uruguay
mforets@gmail.com
[6] University of Illinois Urbana-Champaign, Urbana, IL
yuhaojia98@g.ucla.edu, huan@huan-zhang.com, xiangru4@illinois.edu
[7] Aalborg University, Aalborg, Denmark
christianms@cs.aau.dk

Abstract

This report presents the results of a friendly competition for formal verification of continuous and hybrid systems with artificial intelligence (AI) components. Specifically, machine learning (ML) components in cyber-physical systems (CPS), such as feedforward neural networks used as feedback controllers in closed-loop systems, are considered, which is a class of systems classically known as intelligent control systems, or in more modern and specific terms, neural network control systems (NNCS). We broadly refer to this category as AI and NNCS (AINNCS). The friendly competition took place as part of the workshop Applied Verification for Continuous and Hybrid Systems (ARCH) in 2024. In the 8th edition of this AINNCS category at ARCH-COMP, five tools have been applied to solve 12 benchmarks, which are *CORA*, *CROWN-Reach*, *GoTube*, *JuliaReach*, and *NNV*. This is the year with the largest interest in the community, with two new, and three previous participants. Following last year's trend, despite the additional challenges presented, the verification results have improved year-over-year. In terms of computation time, we can observe that the previous participants have improved as well, showing speed-ups of up to one order of magnitude, such as JuliaReach on the TORA benchmark with ReLU controller, and NNV on the TORA benchmark with both heterogeneous controllers.

1 Introduction

Neural Networks (NNs) have demonstrated an impressive ability to solve complex problems in numerous application domains [78]. The success of these models in contexts such as adaptive control, non-linear system identification [56], image and pattern recognition, function approximation, and machine translation has stimulated the creation of technologies that are directly impacting our everyday lives [68], and has led researchers to believe that these models possess the power to revolutionize a diverse set of arenas [62].

Despite these achievements, there have been reservations about utilizing them within high-assurance systems for various reasons, such as their susceptibility to unexpected and errant behavior caused by slight perturbations in their inputs [43]. In a study by Szegedy et al. [69], the authors demonstrated that carefully applying a hardly perceptible modification to an input image could cause a successfully trained neural network to produce an incorrect classification. These inputs are known as *adversarial examples*, and their discovery has caused concern over the safety, reliability, and security of neural network applications [78]. As a result, some research has been directed toward obtaining an explicit understanding of neural network behavior.

Neural networks are often viewed as "black boxes" whose underlying operation is often incomprehensible. Still, the last several years have witnessed numerous promising white-box verification methods proposed for reasoning about the correctness of their behavior. However, it has been demonstrated that neural network verification is an NP-complete problem [38]. Despite many recent efforts and significant advances in the past decade [55, 73, 74, 75, 7, 23, 24, 39, 46, 67], there are remaining challenges that prevent these approaches from being successfully applied to very large neural networks used in many real-world applications such as [61]. Most of this work also focuses on verifying pre-/post-conditions for neural networks in isolation. Reasoning about their usage behavior in cyber-physical systems, such as in neural network control systems, remains a key challenge.

The following report aims to provide a survey of the landscape of the current capabilities of verification tools for *closed-loop systems with neural network controllers*, as these systems have displayed great utility as a means for learning control laws through techniques such as reinforcement learning and data-driven predictive control [19, 71]. Furthermore, this report aims to provide readers with a perspective on the intellectual progression of this rapidly growing field and stimulate the development of efficient and effective methods capable of use in real-life applications. Since the first iteration, there have been several publications investigating the formal verification of AINNCS, out of which several of them have participated in one or more of the previous competitions such as Verisig [33], VenMas [1] and ReachNN [22], among others [75, 15, 28, 2, 41, 21, 52, 19, 17, 32, 66].

Disclaimer The presented report of the ARCH-COMP friendly competition for *closed-loop systems with neural network controllers*, termed in short AINNCS (Artificial Intelligence / Neural Network Control Systems), aims to provide the landscape of the current capabilities of verification tools for analyzing these systems that are classically known as *intelligent control systems*. This AINNCS ARCH-COMP category is complementary to the ongoing Verification of Neural Networks Competition (VNN-COMP) [9, 55], the latter of which focuses on open-loop specifications of neural networks. In contrast, the AINNCS category focuses on closed-loop behaviors of dynamical systems incorporating neural networks. We want to stress that each tool has unique strengths and not all of the specificities can be highlighted within a single report. To reach a consensus on what benchmarks are used, some

compromises had to be made so that some tools may benefit more from the presented choice than others. To establish further trustworthiness of the results, the code with which the results have been obtained is publicly available at gitlab.com/goranf/ARCH-COMP, and the submitted results are available at arch.repeatability.cps.cit.tum.de/frontend/submissions.

Specifically, this report summarizes results obtained in the 2024 friendly competition of the ARCH workshop[1] for verifying systems of the form

$$\dot{x}(t) = f(x(t), u(x,t)),$$

where $x(t)$ and $u(x,t)$ correspond to the states and inputs of the plant at time t, respectively, and where $u(x,t)$ is the output of a feedforward neural network provided an input of the plant state x at time t. The architecture of the closed-loop systems we consider is depicted in Figure 1, where the input to the neural network controller is additionally sampled.

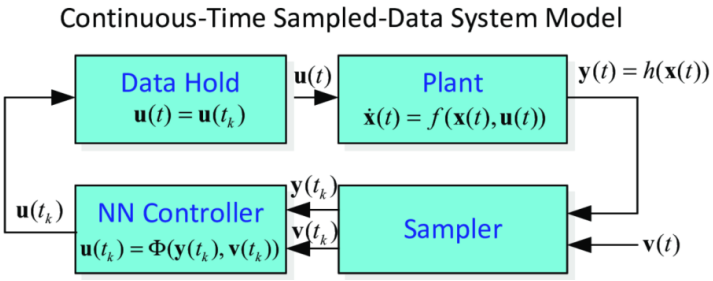


Figure 1: Closed-loop architecture of the benchmarks to be verified.

This year is the 6th iteration of the AINNCS category at ARCH-COMP and builds on the previous iterations and reports [50, 36, 35, 47, 48]. Participating tools are summarized in Sec. 2. Please, see [78] for further details on these and additional tools. The results of our selected benchmark problems are shown in Sec. 3. Similar to last year, we run all tools on the same hardware using docker images for further comparison. The docker images allow an automatic evaluation of the tools on the submission server, thus, giving researchers immediate feedback on the results of their submission. The submission server specifications are given in Appendix A.

The goal of the friendly competition is not to rank the results but rather to present the landscape of existing solutions in a breadth that is impossible with scientific publications in classical venues. Such publications would typically require the presentation of novel techniques, while this report showcases the current state-of-the-art tools. The selection of the benchmarks has been conducted within the forum of the ARCH website (cps-vo.org/group/ARCH), which is visible for registered users, and registration is open for anyone.

[1] Workshop on Applied Verification for Continuous and Hybrid Systems (ARCH), cps-vo.org/group/ARCH

2 Participating Tools

We present a brief overview of all the participating tools in this friendly competition. The tools are *CORA*, *JuliaReach*, and *NNV*, and first-time participants *CROWN-Reach* and *GoTube*. The tools participating in the *Artificial Intelligence / Neural Network Control Systems in Continuous and Hybrid Systems Plants* (AINNCS) category are introduced subsequently in alphabetical order.

CORA *(Tobias Ladner, Matthias Althoff).* CORA [2] is a COntinuous Reachability Analyzer for the formal verification of cyber-physical systems using reachability analysis. It is written in MATLAB and is available at https://cora.in.tum.de. CORA integrates various set representations and operations on them as well as reachability algorithms of various dynamic system classes. For this competition, we used the approach described in [41, 45] for open-loop and closed-loop neural network verification based on polynomial zonotopes [40]. Polynomial zonotopes are particularly well suited for verifying neural networks due to their polynomial time complexity on many operations. CORA realizes a fast layer-based computation of an outer approximation of the output set of networks with various activation functions, including ReLU, sigmoid, and tanh [41], and can automatically refine the neuron abstractions in neural networks to obtain tighter enclosures [45]. Our neural network verification approaches are naturally integrated into our reachability analysis methods for linear and nonlinear plant dynamics. For most benchmarks, we can deploy a fully automatic verification process using CORA: We first simulate random runs to find potential violations of the specifications. If one run violates the specifications, we verify the violating run using reachability analysis, as simulations are not sound. Otherwise, we try to verify the benchmark for the given specifications.

CROWN-Reach *(Xiangru Zhong, Yuhao Jia, Huan Zhang).* CROWN-Reach is a new open-source tool for reachability analysis of neural network control systems developed at UIUC. It aims to strengthen and extend the successful α,β-CROWN neural network verifier [80, 76, 81, 64] to the setting of neural network controller verification. CROWN-Reach consists of four main components: bound-propagation for efficient analysis of neural network controllers, Taylor model for plant analysis, branch-and-bound to refine the reachable set, and a sampling-based falsifier. For the analysis of neural network controllers, we use linear relaxation based perturbation analysis (LiRPA) methods such as CROWN [82] and α-CROWN [80] with extensions to cooperate with Taylor Model flowpipe computation. Our tool is based on the auto_LiRPA library[79], which can automatically compute linear functional over-approximations for neural networks with various activation functions, including ReLU, tanh, and sigmoid, as well as neural networks with general architectures (e.g., residual blocks and custom operators). We use the Flow*[16] library for analyzing the plant with continuous dynamics using Taylor models, and these Taylor models are symbolically combined with the linear bounds from CROWN to form the reachable set of the entire system. The branch-and-bound refinement process splits the input state space and utilizes parallelization (including both GPU and CPU) to achieve quick and precise analysis. The bound propagation process can be accelerated on GPUs and can scale to very large networks, while the computation of Taylor models is executed on CPUs using multi-threading. A paper describing the algorithm details of CROWN-Reach is currently being prepared. Code is available at https://github.com/Verified-Intelligence/CROWN-Reach.

GoTube v2.0 *(DatenVorsprung GmbH).* GoTube v2.0 is a formal statistical verification algorithm for Neural-Network-Controlled Systems that uses reachability analysis to formally quantifiy the behavioral robustness and safety of any time-continuous process. The algorithm solves a set of global optimization (Go) problems over a given time horizon to construct a tight enclosure (Tube) of the set of all process executions starting from a ball of initial states. As an extension of GoTube [30, 29], GoTube v2.0 is able to handle any kind and any amount of neural network controllers within a nonlinear continuous system (cyber-physical system), whereas the first version was only able to handle continuous-depth neural networks and control systems with one controller. In addition, GoTube v2.0 includes verification methods while GoTube was only able to perform reachability analysis. GoTube v2.0 is able to perform a fully automatic verification process for all benchmarks. Through its construction, GoTube ensures that the bounding tube is conservative up to a desired confidence and up to a desired tightness factor. GoTube can trade runtime for confidence, so by setting a lower confidence it is possible to get a result faster. If the result of that fast run with low confidence is "falsified" or "unknown", the analysis is finished, as we will also not be able to verify the result with a higher confidence. Only if the result is "verified", we will need to increase the confidence until the desired confidence of the verification result is reached. Compared to advanced reachability analysis tools for Neural-Network-Controlled Systems, GoTube does not accumulate over-approximation errors between time steps and avoids the infamous wrapping effect inherent in symbolic techniques. Thus, it is able to handle bigger initial radii, systems with bigger differences between trajectories (e.g. Cartpole benchmark) as well as time horizons well beyond what has been previously possible.

JuliaReach *(Luis Benet, Marcelo Forets, Christian Schilling).* JuliaReach [15] is an open-source software suite for reachability computations of dynamical systems, written in the Julia language and available at `http://github.com/JuliaReach`. The package ClosedLoopReachability.jl handles the closed-loop analysis and queries sub-problems to our other libraries ReachabilityAnalysis.jl for continuous-time analysis of plant models and NeuralNetworkReachability.jl for set propagation through neural networks (both forward and backward [27]). Additional set computations are performed with LazySets.jl [26]. The algorithm we use is described in [63]. For the plant analysis, we use the sound algorithm `TMJets` based on interval arithmetic and Taylor models; this algorithm is implemented in TaylorModels.jl [11, 14], which itself incorporates TaylorSeries.jl [12, 13] and TaylorIntegration.jl [57]. The algorithm uses a jet transportation of a Taylor polynomial with interval coefficients. It has the following main parameters for tweaking: the absolute tolerance `abstol` and two parameters to define the order at which the Taylor expansion is cut in time (`orderT`) resp. in space (`orderQ`). For the neural-network analysis, we use an abstract interpretation based on zonotopes [67]. For falsification, we choose an initial point but then use set-based analysis for validated simulations.

NNV *(Diego Manzanas Lopez, Taylor Johnson).* The Neural Network Verification (NNV) Tool [75, 49] is a formal verification software tool for deep learning models and cyber-physical systems with neural network components written in MATLAB and available at `https://github.com/verivital/nnv`. NNV uses a star-set state-space representation and reachability algorithm that allows for a layer-by-layer computation of exact or overapproximate reachable sets for feed-forward [73], convolutional [70], semantic segmentation (SSNN) [74], and recurrent (RNN)[72] neural networks, as well as neural network control systems (NNCS) [71, 75] and neural ordinary differential equations (Neural ODEs) [53]. The star-set based algorithm is naturally parallelizable, which allows NNV to be designed to perform efficiently on multi-

core platforms. Additionally, if a particular safety property is violated, NNV can be used to construct and visualize the complete set of counterexample inputs for a neural network (exact-analysis). Using NNV in combination with HyST [8, 6] and CORA [2, 3, 4] allows for the verification of closed-loop neural network control systems with nonlinear plant dynamics.

3 Benchmarks

We have selected 12 benchmarks for this year's competition – we reuse the same benchmarks (with modification on the specifications of 3 of them) and add two new ones, described at the end of the section [48]. A few of them, such as the TORA benchmark, are presented with several different controllers to be analyzed. We now describe these benchmarks in no particular order and have made them readily available online.[2] All benchmarks are derived for continuous time. Given the continuous dynamics $\dot{x} = f(x)$, where $x \in \mathbb{R}^n$ is the state vector, the discrete-time versions for a time increment of Δt are obtained in this competition using forward Euler integration:

$$x(k+1) = x(k) + f(x)\Delta t.$$

3.1 Adaptive Cruise Controller (ACC)

The Adaptive Cruise Control (ACC) benchmark is a system that tracks a set velocity and maintains a safe distance from a lead vehicle by adjusting the longitudinal acceleration of an ego vehicle [54]. The neural network computes optimal control actions while satisfying safe distance, velocity, and acceleration constraints using model predictive control (MPC) [59]. For this case study, the ego car is set to travel at a set speed $v_{set} = 30$ and maintains a safe distance D_{safe} from the lead car. The car's dynamics are described by the following equations [71, p. 17]:

$$\begin{aligned} \dot{x}_{\text{lead}}(t) = v_{\text{lead}}(t), \dot{v}_{\text{lead}}(t) = a_{\text{lead}}(t), \dot{a}_{\text{lead}}(t) = -2a_{\text{lead}}(t) + 2a_{c,\text{lead}} - uv^2_{\text{lead}}(t), \\ \dot{x}_{\text{ego}}(t) = v_{\text{ego}}(t), \dot{v}_{\text{ego}}(t) = a_{\text{ego}}(t), \dot{a}_{\text{ego}}(t) = -2a_{\text{ego}}(t) + 2a_{c,\text{ego}} - uv^2_{\text{ego}}(t), \end{aligned} \tag{1}$$

where x_i is the position, v_i is the velocity, a_i is the acceleration of the car, $a_{c,i}$ is the acceleration control input applied to the car, and $u = 0.0001$ is a coefficient for air drag, where $i \in \{\text{ego, lead}\}$. We evaluate a neural network controller with five layers and 20 neurons each for this benchmark. The inputs of the controller are the set speed v_{set}, the desired time gap T_{gap}, the ego velocity v_{ego}, the distance $D_{\text{rel}} = x_{\text{lead}} - x_{\text{ego}}$, as well as the relative velocity v_{rel}, and the output is $a_{\text{c,ego}}$.

Specifications The verification objective of this system is that given a scenario where both cars are driving safely, the lead car suddenly slows down with $a_{c,\text{lead}}$ = -2. We want to check whether there is a collision in the following 5 s. Formally, this safety specification of the system can be expressed as $D_{\text{rel}} \geq D_{\text{safe}}$, where $D_{\text{safe}} = D_{\text{default}} + T_{\text{gap}} \cdot v_{\text{ego}}$, and $T_{\text{gap}} = 1.4$ s and $D_{\text{default}} = 10$. The initial conditions are: $x_{\text{lead}}(0) \in$ [90,110], $v_{\text{lead}}(0) \in$ [32,32.2], $a_{\text{lead}}(0) = a_{\text{ego}}(0) = 0$, $v_{\text{ego}}(0) \in$ [30, 30.2], $x_{\text{ego}} \in$ [10,11]. A control period of 0.1 s is used.

[2] GitHub repository of benchmarks: https://github.com/verivital/ARCH-COMP2024

3.2 TORA

This benchmark considers translational oscillations by a rotational actuator (TORA) [19, 34], where a cart is attached to a wall with a spring and is free to move on a friction-less surface. The cart has a weight attached to an arm inside it, which is free to rotate about an axis. This serves as the control input to stabilize the cart at $\mathbf{x} = \mathbf{0}$. The model is a four-dimensional system, given by the following equations [34, eq. (4)]:

$$\dot{x_1} = x_2, \quad \dot{x_2} = -x_1 + 0.1\sin(x_3), \quad \dot{x_3} = x_4, \quad \dot{x_4} = u. \tag{2}$$

This benchmark has three neural network controllers: the first has three ReLU hidden layers and a linear output layer. This controller was trained using a data-driven model predictive controller proposed in [20]. Note that the output of the neural network $f(\mathbf{x})$ needs to be normalized to obtain u, namely $u = f(\mathbf{x}) - 10$. The sampling time for this controller is 1 s, and we verify it against specification 1 below. The other two controllers have three hidden layers of 20 neurons each and one output layer. In contrast to the first controller, we use sigmoid activation functions for the hidden layers and a tanh output layer. The sampling time of these controllers is 0.5 s, the output of the neural network $f(\mathbf{x})$ needs to be post-processed as $u = 11 \cdot f(\mathbf{x})$, and we verify them against specification 2 below.

Specification 1. This is a safety specification. For an initial set of $x_1 \in [0.6, 0.7]$, $x_2 \in [-0.7, -0.6]$, $x_3 \in [-0.4, -0.3]$, and $x_4 \in [0.5, 0.6]$, the system states have to stay within the box $\mathbf{x} \in [-2, 2]^4$ for a time window of 20 s.

Specification 2. For an initial set of $x_1 \in$ [-0.77, -0.75], $x_2 \in$ [-0.45, -0.43], $x_3 \in$ [0.51, 0.54], and $x_4 \in [-0.3, -0.28]$, it is required that the system reaches the set $x_1 \in [-0.1, 0.2]$, $x_2 \in [-0.9, -0.6]$ within a time window of 5 s.

3.3 Unicycle

This benchmark considers a unicycle model of a car [19] with the x and y coordinates on a two-dimensional plane, the velocity magnitude (speed), and steering angle as state variables. The dynamic equations are (see [5, Sec. III.B]; a different input is used here):

$$\dot{x}_1 = x_4\cos(x_3), \quad \dot{x}_2 = x_4\sin(x_3), \quad \dot{x}_3 = u_2, \quad \dot{x}_4 = u_1 + w, \tag{3}$$

where w is a bounded error in the range $10^{-4}[-1, 1]$. A neural network controller was trained for this system using a model predictive controller as a "demonstrator" or "teacher". The trained network has one hidden layer with 500 neurons. Note that the output of the neural network $f(\mathbf{x})$ needs to be normalized in order to obtain (u_1, u_2), namely $u_i = f(\mathbf{x})_i - 20$. The sampling time for this controller is 0.2 s.

Specification This is a reachability specification. For an initial set of $x_1 \in [9.5, 9.55]$, $x_2 \in [-4.5, -4.45]$, $x_3 \in [2.1, 2.11]$, and $x_4 \in [1.5, 1.51]$, the system has to reach the set $x_1 \in [-0.6, 0.6], x_2 \in [-0.2, 0.2], x_3 \in [-0.06, 0.06], x_4 \in [-0.3, 0.3]$ within a time window of 10 s.

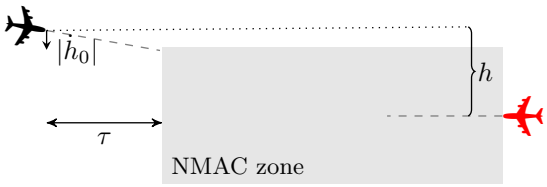


Figure 2: VerticalCAS encounter geometry

3.4 VerticalCAS

This benchmark is a closed-loop variant of the *aircraft collision avoidance system ACAS X*. The scenario involves two aircraft, the ownship and the intruder, where the ownship is equipped with a collision avoidance system referred to as VerticalCAS [37]. VerticalCAS issues vertical climb rate advisories every second to the ownship pilot to avoid a near mid-air collision (NMAC). Near mid-air collisions are regions where the ownship and the intruder are separated by less than 100ft vertically and 500ft horizontally. The ownship (black) is assumed to have a constant horizontal speed, and the intruder (red) is assumed to follow a constant horizontal trajectory towards ownship, see Figure 2. The current geometry of the system is described by

- h, intruder altitude relative to ownship,
- $\dot{h}_0$, ownship vertical climb rate, and
- τ, the time until the ownship (black) and intruder (red) are no longer horizontally separated.

We can, therefore, assume that the intruder is static and the horizontal separation τ decreases by one each second. There are nine advisories and each of them instructs the pilot to accelerate until the vertical climb rate of the ownship complies with the advisory:

1. COC: Clear Of Conflict;
2. DNC: Do Not Climb;
3. DND: Do Not Descend;
4. DES1500: Descend at least 1500 ft/s;
5. CL1500: Climb at least 1500 ft/s;
6. SDES1500: Strengthen Descent to at least 1500 ft/s;
7. SCL1500: Strengthen Climb to at least 1500 ft/s;
8. SDES2500: Strengthen Descent to at least 2500 ft/s;
9. SCL2500: Strengthen Climb to at least 2500 ft/s.

In addition to the parameters describing the geometry of the encounter, the current state of the system stores the advisory adv $\in \{1, \ldots, 9\}$ (numbers correspond to the above list) issued to the ownship at the previous time step. VerticalCAS is implemented as nine ReLU networks N_i, one for each (previous) advisory, with three inputs $(h, \dot{h}_0, \tau)$, five fully-connected hidden layers of 20 units each, and nine outputs representing the score of each possible advisory. Therefore, given a current state $(h, \dot{h}_0, \tau, \text{adv})$, the new advisory adv$'$ is obtained by computing the argmax of the output of N_{adv} on $(h, \dot{h}_0, \tau)$.

Given the new advisory, the pilot can choose acceleration $\ddot{h}_0$ as follows. If the new advisory is COC, then it can be any acceleration from the set $\{-\frac{g}{8}, 0, \frac{g}{8}\}$. For all remaining advisories, if the previous advisory coincides with the new one and the current climb rate complies with the new advisory (e.g., $\dot{h}_0$ is non-positive for DNC and $\dot{h}_0 \geq 1500$ for CL1500), the acceleration $\ddot{h}_0$ is 0; otherwise, the pilot can choose any acceleration $\ddot{h}_0$ from the given sets:

- DNC: $\{-\frac{g}{3}, -\frac{7g}{24}, -\frac{g}{4}\}$;
- DND: $\{\frac{g}{4}, \frac{7g}{24}, \frac{g}{3}\}$;
- DES1500: $\{-\frac{g}{3}, -\frac{7g}{24}, -\frac{g}{4}\}$;
- CL1500: $\{\frac{g}{4}, \frac{7g}{24}, \frac{g}{3}\}$;
- SDES1500: $\{-\frac{g}{3}\}$;
- SCL1500: $\{\frac{g}{3}\}$;
- SDES2500: $\{-\frac{g}{3}\}$;
- SCL2500: $\{\frac{g}{3}\}$,

where g represents the gravitational constant 32.2 ft/s^2.

It was proposed to tweak the benchmark for the tools that cannot efficiently account for all possible acceleration choices. Those tools can consider two strategies for picking a single acceleration at each time step:

- a worst-case scenario selection, where we choose the acceleration to take the ownship closer to the intruder.
- always select the acceleration in the middle.

Given the current system state $(h, \dot{h}_0, \tau, \text{adv})$, the new advisory adv$'$ and the acceleration $\ddot{h}_0$, the new state of the system can be computed by the following equations [37, eq. (15)]:

$$\begin{aligned} h(k+1) &= h(k) - \dot{h}_0(k)\Delta\tau - 0.5\ddot{h}_0(k)\Delta\tau^2 \\ \dot{h}_0(k+1) &= \dot{h}_0(k) + \ddot{h}_0(k)\Delta\tau \\ \tau(k+1) &= \tau(k) + \Delta\tau \\ \text{adv}(k+1) &= \text{adv}' \end{aligned}$$

where $\Delta\tau = 1$.

Specification The ownship has to be outside of the NMAC zone after $k \in \{1, \ldots, 10\}$ time steps, i.e., $h(k) > 100$ or $h(k) < -100$, for all possible choices of acceleration by the pilot. The set of initial states considered is: $h(0) \in [-133, -129]$, $\dot{h}_0(0) \in \{-19.5, -22.5, -25.5, -28.5\}$, $\tau(0) = 25$ and adv(0) = COC.

3.5 Single Pendulum

We consider a classical inverted pendulum. A ball of mass m is attached to a massless beam of length L. The beam is actuated with a torque T and we assume viscous friction with a friction coefficient of c. The governing equation of motion can be obtained as [51, eq. (1)]:

$$\ddot{\theta} = \frac{g}{L}\sin\theta + \frac{1}{mL^2}\left(T - c\,\dot{\theta}\right), \tag{4}$$

where θ is the angle of the link concerning the upward vertical axis and $\dot{\theta}$ is the angular velocity. After defining the state variables $x_1 = \theta$ and $x_2 = \dot{\theta}$, the dynamics in state-space form is

$$\dot{x}_1 = x_2, \tag{5a}$$

$$\dot{x}_2 = \frac{g}{L}\sin x_1 + \frac{1}{mL^2}\left(T - c\, x_2\right). \tag{5b}$$

Controllers are trained using *behavior cloning*, a supervised learning approach for training controllers. The code, as well as training procedures, are provided. The model parameters are chosen as

$$m = 0.5, L = 0.5, c = 0, g = 1, \tag{6}$$

and the time step for the controller and the discrete-time model is $\Delta t = 0.05$. The initial set is

$$x \in [1.0, 1.175] \times [0.0, 0.2].$$

Specification $\forall t \in [0.5, 1] : \theta \in [0, 1]$ (analogously for $k \in [10, 20]$ in discrete time).

3.6 Double Pendulum

The double pendulum is an inverted two-link pendulum with equal point masses m at the end of connected mass-less links of length L. The links are actuated with torques T_1 and T_2, and we assume viscous friction exists with a coefficient of c. The governing equations of motion are described by the following equations [51, eq. (3a-b)]:

$$2\ddot{\theta}_1 + \ddot{\theta}_2\cos(\theta_2 - \theta_1) - \dot{\theta}_2^2\sin(\theta_2 - \theta_1) - 2\frac{g}{L}\sin\theta_1 + \frac{c}{mL^2}\dot{\theta}_1 = \frac{1}{mL^2}T_1, \tag{7a}$$

$$\ddot{\theta}_1\cos(\theta_2 - \theta_1) + \ddot{\theta}_2 + \dot{\theta}_1^2\sin(\theta_2 - \theta_1) - \frac{g}{L}\sin\theta_2 + \frac{c}{mL^2}\dot{\theta}_2 = \frac{1}{mL^2}T_2, \tag{7b}$$

where θ_1 and θ_2 are the angles of the links concerning the upward vertical axis (see Figure 3) and g is the gravitational acceleration. After defining the state vector as $x = [\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2]^T$, the dynamics in state-space form is

$$\dot{x}_1 = x_3, \tag{8a}$$

$$\dot{x}_2 = x_4, \tag{8b}$$

$$\dot{x}_3 = \frac{1}{2\left(\frac{\cos^2(x_1 - x_2)}{2} - 1\right)}\cos(x_1 - x_2)\left({x_3}^2\sin(x_1 - x_2) - \cos(x_1 - x_2)\left(\frac{g\sin(x_1)}{L}\right.\right. \tag{8c}$$

$$\left. - \frac{{x_4}^2\sin(x_1 - x_2)}{2} + \frac{T_1 - c\,x_3}{2\,L^2\,m}\right) + \frac{g\sin(x_2)}{L} + \frac{T_2 - c\,x_4}{L^2\,m}\Bigg) \tag{8d}$$

$$- \frac{{x_4}^2\sin(x_1 - x_2)}{2} + \frac{g\sin(x_1)}{L} + \frac{T_1 - c\,x_3}{2\,L^2\,m}, \tag{8e}$$

$$\dot{x}_4 = \frac{-1}{\frac{\cos^2(x_1 - x_2)}{2} - 1}\left({x_3}^2\sin(x_1 - x_2) - \cos(x_1 - x_2)\left(\frac{g\sin(x_1)}{L} - \frac{{x_4}^2\sin(x_1 - x_2)}{2}\right.\right. \tag{8f}$$

$$\left. + \frac{T_1 - c\,x_3}{2\,L^2\,m}\right) + \frac{g\sin(x_2)}{L} + \frac{T_2 - c\,x_4}{L^2\,m}\Bigg). \tag{8g}$$

The controllers for the double pendulum benchmark are obtained using the same methods as the controllers for the single pendulum benchmark; also here, the code as well as training procedures are provided. The model parameters are chosen as in (6) The initial set is

$$x \in [1.0, 1.3]^4.$$

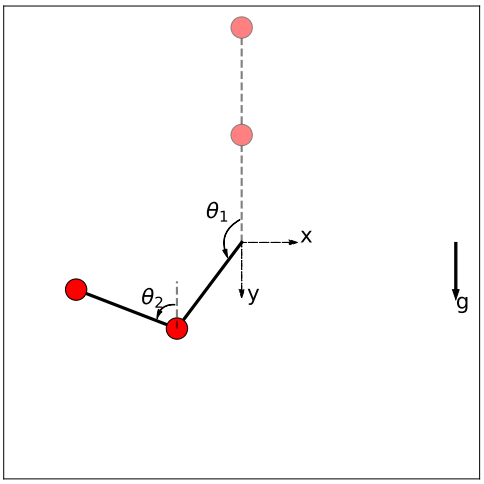


Figure 3: Inverted double pendulum. The goal is to keep the pendulum upright (dashed schematics)

Specification 1 $\forall t \in [0,1] : x \in [-1.7, 2]^4$ (analogously for $k \in [0, 20]$ in discrete time) for $\Delta t = 0.05$.

Specification 2 $\forall t \in [0, 0.4] : x \in [-1.5, 1.5]^4$ (analogously for $k \in [0, 20]$ in discrete time) for $\Delta t = 0.02$.

We verify *controller double pendulum less robust* against specification 1 and *controller double pendulum more robust* against specification 2.

3.7 Airplane

The airplane example consists of a dynamical system that is a simple model of a flying airplane as shown in Figure 4. The state is

$$x = [s_x, s_y, s_z, v_x, v_y, v_z, \phi, \theta, \psi, r, p, q]^T, \tag{9}$$

where (s_x, s_y, s_z) is the position of the center of gravity, (v_x, v_y, v_z) are the components of velocity in (x, y, z) directions, (p, q, r) are body rotation rates, and (ϕ, θ, ψ) are the Euler angles. The equations of motion are reduced to [51, eq. (7)]:

$$\dot{v}_x = -g \sin\theta + \frac{F_x}{m} - qv_z + rv_y, \tag{10a}$$

$$\dot{v}_y = g \cos\theta \sin\phi + \frac{F_y}{m} - rv_x + pv_z, \tag{10b}$$

$$\dot{v}_z = g \cos\theta \cos\phi + \frac{F_z}{m} - pv_y + qv_x, \tag{10c}$$

$$I_x\dot{p} + I_{xz}\dot{r} = M_x - (I_z - I_y)qr - I_{xz}pq, \tag{10d}$$

$$I_y\dot{q} = M_y - I_{xz}\left(r^2 - p^2\right) - (I_x - I_z)pr, \tag{10e}$$

$$I_{xz}\dot{p} + I_z\dot{r} = M_z - (I_y - I_x)qp - I_{xz}rq. \tag{10f}$$

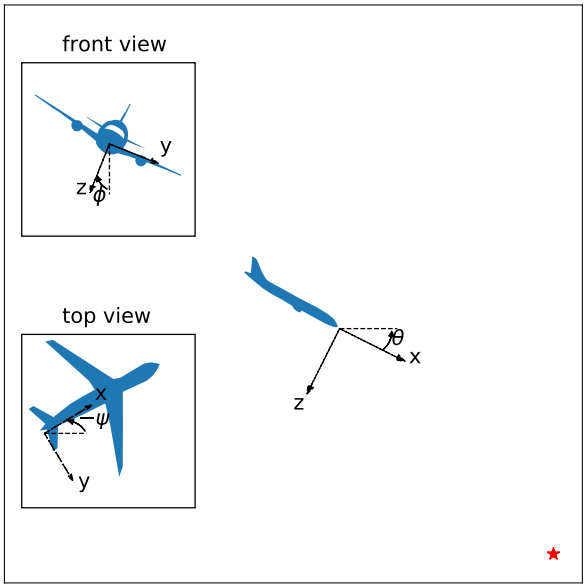


Figure 4: The airplane example.

The mass of the airplane is denoted with m and I_x, I_y, I_z and I_{xz} are the moment of inertia with respect to the indicated axis; see Figure 4. The control parameters include three force components F_x, F_y and F_z and three moment components M_x, M_y, M_z. Note that for simplicity, we assume that the aerodynamic forces are absorbed in the force vector F. In addition to these six equations, we have six additional kinematic equations [51, eq. (8,9)]:

$$\begin{bmatrix} \dot{s}_x \\ \dot{s}_y \\ \dot{s}_z \end{bmatrix} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \tag{11}$$

and

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} 1 & \tan\theta\sin\phi & \tan\theta\cos\phi \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sec\theta\sin\phi & \sec\theta\cos\phi \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}. \tag{12}$$

As in the pendulum benchmarks, controllers are trained for the airplane problem using behavior cloning. The system involves the model parameters

$$m = 1, I_x = I_y = I_z = 1, I_{xz} = 0, g = 1,$$

and the time step for the controller and the discrete-time model is $\Delta t = 0.1$. The initial set is

$$x = y = z = r = p = q = 0, \quad [v_x, v_y, v_z, \phi, \theta, \psi] \in [0.0, 1.0]^6.$$

Specification $\forall t \in [0, 2] : s_y \in [-1, 1]$, $[\phi, \theta, \psi] \in [-1.0, 1.0]^3$. Analogously for $k \in [0, 20]$ in discrete time.

We verify the airplane controller against the specification above.

3.8 Attitude Control

We consider the attitude control of a rigid body with six states and three inputs [58, 65]. The system dynamics is given by [58, Sec. V]:

$$\begin{aligned}
&\dot{\omega}_1 = 0.25(u_0 + \omega_2\omega_3), \qquad \dot{\omega}_2 = 0.5(u_1 - 3\omega_1\omega_3), \qquad \dot{\omega}_3 = u_2 + 2\omega_1\omega_2,\\
&\dot{\psi}_1 = 0.5\left(\omega_2(\psi_1^2+\psi_2^2+\psi_3^2-\psi_3)+\omega_3(\psi_1^2+\psi_2^2+\psi_2+\psi_3^2)+\omega_1(\psi_1^2+\psi_2^2+\psi_3^2+1)\right),\\
&\dot{\psi}_2 = 0.5\left(\omega_1(\psi_1^2+\psi_2^2+\psi_3^2+\psi_3)+\omega_3(\psi_1^2-\psi_1+\psi_2^2+\psi_3^2)+\omega_2(\psi_1^2+\psi_2^2+\psi_3^2+1)\right),\\
&\dot{\psi}_3 = 0.5\left(\omega_1(\psi_1^2+\psi_2^2-\psi_2+\psi_3^2)+\omega_2(\psi_1^2+\psi_1+\psi_2^2+\psi_3^2)+\omega_3(\psi_1^2+\psi_2^2+\psi_3^2+1)\right),
\end{aligned}$$

wherein the state $x = (\omega^T, \psi^T)^T$ consists of the angular velocity vector in a body-fixed frame $\omega \in \mathbb{R}^3$ and the Rodrigues parameter vector $\psi \in \mathbb{R}^3$.

The control torque $u \in \mathbb{R}^3$ is updated every 0.1 s by a neural network with three hidden layers, each of which has 64 neurons. The activations of the hidden layers are sigmoid and identity, respectively. We train the neural-network controller using supervised learning methods to learn from a known nonlinear controller [58]. The initial state set is:

$$\begin{aligned}
&\omega_1 \in [-0.45, -0.44], \omega_2 \in [-0.55, -0.54], \omega_3 \in [0.65, 0.66],\\
&\psi_1 \in [-0.75, -0.74], \psi_2 \in [0.85, 0.86], \psi_3 \in [-0.65, -0.64].
\end{aligned}$$

Specification The system should not reach the following unsafe set in 3 s (30 time steps):

$$\begin{aligned}
&\omega_1 \in [-0.2, 0], \omega_2 \in [-0.5, -0.4], \omega_3 \in [0, 0.2],\\
&\psi_1 \in [-0.7, -0.6], \psi_2 \in [0.7, 0.8], \psi_3 \in [-0.4, -0.2].
\end{aligned}$$

We want to show that the above specification does not hold.

3.9 Quadrotor

This benchmark studies a neural-network controlled quadrotor (QUAD) with twelve state variables [10]. We have the inertial (north) position x_1, the inertial (east) position x_2, the altitude x_3, the longitudinal velocity x_4, the lateral velocity x_5, the vertical velocity x_6, the roll angle x_7, the pitch angle x_8, the yaw angle x_9, the roll rate x_{10}, the pitch rate x_{11}, and the yaw rate x_{12}. The control torque $u \in \mathbb{R}^3$ is updated every 0.1 s by a neural network with 3 hidden layers, each of which has 64 neurons. The activations of the hidden layers and the output layer are sigmoid and identity, respectively. The dynamics are given by the following equations [10, eq. (12-16)]:

$$
\begin{aligned}
\dot{x}_1 =& \cos(x_8)\cos(x_9)x_4 + (\sin(x_7)\sin(x_8)\cos(x_9) - \cos(x_7)\sin(x_9))\,x_5 \\
&+ (\cos(x_7)\sin(x_8)\cos(x_9) + \sin(x_7)\sin(x_9))\,x_6, \\
\dot{x}_2 =& \cos(x_8)\sin(x_9)x_4 + (\sin(x_7)\sin(x_8)\sin(x_9) + \cos(x_7)\cos(x_9))\,x_5 \\
&+ (\cos(x_7)\sin(x_8)\sin(x_9) - \sin(x_7)\cos(x_9))\,x_6, \\
\dot{x}_3 =& \sin(x_8)x_4 - \sin(x_7)\cos(x_8)x_5 - \cos(x_7)\cos(x_8)x_6, \\
\dot{x}_4 =& x_{12}x_5 - x_{11}x_6 - g\sin(x_8,) \\
\dot{x}_5 =& x_{10}x_6 - x_{12}x_4 + g\cos(x_8)\sin(x_7), \\
\dot{x}_6 =& x_{11}x_4 - x_{10}x_5 + g\cos(x_8)\cos(x_7) - g - u_1/m, \\
\dot{x}_7 =& x_{10} + \sin(x_7)\tan(x_8)x_{11} + \cos(x_7)\tan(x_8)x_{12}, \\
\dot{x}_8 =& \cos(x_7)x_{11} - \sin(x_7)x_{12}, \\
\dot{x}_9 =& \frac{\sin(x_7)}{\cos(x_8)}x_{11} - \frac{\cos(x_7)}{\cos(x_8)}x_{12}, \\
\dot{x}_{10} =& \frac{J_y - J_z}{J_x}x_{11}x_{12} + \frac{1}{J_x}u_2, \\
\dot{x}_{11} =& \frac{J_z - J_x}{J_y}x_{10}x_{12} + \frac{1}{J_y}u_3, \\
\dot{x}_{12} =& \frac{J_x - J_y}{J_z}x_{10}x_{11} + \frac{1}{J_z}\tau_\psi,
\end{aligned}
$$

where

$$
\begin{aligned}
&g = 9.81, \quad m = 1.4, \quad J_x = 0.054, \\
&J_y = 0.054, \quad J_z = 0.104, \quad \tau_\psi = 0.
\end{aligned}
$$

The initial set is:

$$
\begin{aligned}
&x_1 \in [-0.4, 0.4], x_2 \in [-0.4, 0.4], x_3 \in [-0.4, 0.4], x_4 \in [-0.4, 0.4], \\
&x_5 \in [-0.4, 0.4], x_6 \in [-0.4, 0.4], x_7 = 0, x_8 = 0, x_9 = 0, x_{10} = 0, x_{11} = 0, x_{12} = 0.
\end{aligned}
$$

Specification The control goal is to stabilize the attitude x_3 to a goal region $[0.94, 1.06]$ and remain within these bounds with a time horizon of 5 s (50 time steps).

3.10 2D Spacecraft Docking

In the 2D spacecraft docking environment, the state of an active deputy spacecraft is expressed relative to the passive chief spacecraft in Hill's reference frame [31]. The dynamics are given by a first-order approximation of the relative motion dynamics between the deputy and chief spacecraft, which is given by Clohessy-Wiltshire [18] equations [60, eq. (12)],

$$
\begin{bmatrix} \dot{s}_x \\ \dot{s}_y \\ \ddot{s}_x \\ \ddot{s}_y \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 3n^2 & 0 & 0 & 2n \\ 0 & 0 & -2n & 0 \end{bmatrix} \begin{bmatrix} s_x \\ s_y \\ \dot{s}_x \\ \dot{s}_y \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{1}{m} & 0 \\ 0 & \frac{1}{m} \end{bmatrix} u, \tag{13}
$$

where $m = 12$ (kg), $n = 0.001027$ (rad/s), and $u \in \mathbb{R}^2$.

The neural network controller was trained on the Docking 2D environment with reinforcement learning using the training procedure described in [60]. However, the training procedure differed in providing only the full state (position and velocity) as input and with hard clipping of output actions replaced with soft tanh clipping. The neural network architecture was a shallow multilayer perceptron with 2 hidden layers of 256 neurons and tanh activation functions, and a linear output layer. The pre-processing and post-processing of the controller have been incorporated into the model as linear layers. The controller was trained with a sampling time of 1 s.

Specification The spacecraft should satisfy the following safety constraints for 40 s:

$$(\dot{s}_x^2 + \dot{s}_y^2)^{\frac{1}{2}} \leq 0.2 + 2n(s_x^2 + s_y^2)^{\frac{1}{2}}, \tag{14}$$

given the initial set is

$$s_x \in [70, 106], s_y \in [70, 106], \dot{s}_x \in [-0.28, 0.28], \dot{s}_y \in [-0.28, 0.28].$$

3.11 Navigation Task

The navigation benchmark models a simplified model of a robot [19] navigating to a goal region while avoiding an obstacle along its path. The state is four-dimensional consisting of the horizontal and vertical position x, y, the angle θ of the robot, and velocity ν. The controller gets all states as input and has an output $u \in [-1, 1]^2$. The dynamics of the system are given by:

$$\dot{s} = \begin{bmatrix} \nu \cos\theta \\ \nu \sin\theta \\ u_{(1)} \\ u_{(2)} \end{bmatrix}. \tag{15}$$

This benchmark is designed to compare how different training schemes improve the verifiability of a controller. As a baseline, the first controller was trained using standard (point-based) reinforcement learning. We used adversarial training to obtain a second, more robust controller: During training, uncertainties of a given state are modeled using sets, and the weights are updated based on the entire input set rather than individual points. This approach was first developed in the supervised learning setting [42], and has also been applied during reinforcement learning [77]. Both networks have two hidden layers with 64 neurons each and ReLU activation, with a final layer with tanh activation to match the benchmark description. The controllers were trained with a sampling time of 0.2 s.

Specification The robot should avoid an obstacle during the entire time horizon ($t \in [0, 6]$s) and reach the goal region at $t = 6$s. The initial set is given by:

$$x \in [2.9, 3.1],\ y \in [2.9, 3.1],\ \theta \in [0, 0],\ \nu \in [0, 0].$$

The obstacle is located at:

$$x \in [1, 2],\ y \in [1, 2],\ \theta \in [-\infty, \infty],\ \nu \in [-\infty, \infty].$$

The goal region is located at:

$$x \in [-0.5, 0.5],\ y \in [-0.5, 0.5],\ \theta \in [-\infty, \infty],\ \nu \in [-\infty, \infty].$$

3.12 CartPole

We consider an inverted pendulum (pole) mounted on a movable one-directional cart. The controller's goal is to balance the pendulum upright by moving the cart. The pole of length l_p has a total mass m_p. The cart's responsiveness is defined by its acceleration magnitude x_{acc}. By using cartpole equations [25] and the momentum inertia defined by $I = \frac{3}{4} m_p l^2$ we get a physical description of the system. With $x_1 \in [-0.5, 0.5]$ being the carts position (with its velocity $x_2 = \dot{x}_1$) and $x_3 \in \mathbb{R}$ being the angle of the pendulum (with $\{2\pi z, z \in \mathbb{Z}\}$ being the inverted state, $x_4 = \dot{x}_3$ the angular velocity) the state vector is defined by $[x_1, x_2, x_3, x_4]^T$. This results into the following behaviour.

$$\dot{x}_1 = x_2 \tag{16}$$
$$\dot{x}_2 = x_{acc} \cdot f(x_1, x_2, \sin x_3, \cos x_3, x_4) \tag{17}$$
$$\dot{x}_3 = x_4 \tag{18}$$
$$\dot{x}_4 = \frac{m_p \cdot l_p \cdot (g \cdot \sin x_3 - x_{acc} \cdot f(x_1, x_2, \sin x_3, \cos x_3, x_4) \cdot \cos x_3) - \mu_p \cdot x_4}{I} \tag{19}$$

with $f(x_1, x_2, \sin x_3, \cos x_3, x_4)$ being the output of the controller.
The system parameters are given by

$$l_p = 0.41\,m, \quad m_p = 0.08\,kg, \quad I = 10.5 \cdot 10^{-3}\,kg\,m^2,$$
$$x_{acc} = 2\,\frac{m}{s^2}, \quad \mu_p = 2.1 \cdot 10^{-3}\,\frac{kg\,m^2}{s^2}, \quad g = 9.8\,\frac{m}{s^1}$$

The initial set is defined by

$$(x_1, x_2, x_3, x_4) \in [-0.1, 0.1] \times [-0.05, 0.05] \times [-0.1, 0.1] \times [-0.05, 0.05]$$

The controller has a latency of $\tau = 0.02\,s$.

Specification After a settling time of 8 seconds the controller is able to stabilize the pendulum ($(x_3, x_4 \in [-0.001, 0.001]^2)$) in the middle of the rail ($x_1 \in [-0.001, 0.001]$):

$$\forall t \in [8\,s, 10\,s]: \quad x_1, x_3, x_4 \in [-0.001, 0.001]^3$$

4 Verification Results

For each of the participating tools, we obtained verification results for some or all of the benchmarks. This year's competition (as in 2023) included the submission of the tools for repeatability prior to the writing of the report to ensure a fairer competition. Reachable sets are shown for those methods that are able to construct them. The published results are available on the website of the submission system[3] and in the repeatability repository[4].

4.1 CORA

This subsection presents the results of *CORA* computed with the automatic verification algorithm described in Sec. 2 if not stated otherwise. The exact parameters can be found in the repeatability repository. An overview of the results is given in Tab. 1.

[3]Submission system: https://arch.repeatability.cps.cit.tum.de/frontend
[4]Repeatability repository: https://gitlab.com/goranf/ARCH-COMP/-/tree/master/2024/AINNCS

Table 1: ***CORA.*** Overview of results: Verified (✓), falsified (✗), and unknown (?).

Benchmark	Instance	Result	Time [s]
Unicycle	reach	✓	7.910
ACC	safe-distance	✓	4.720
TORA	remain	✓	15.030
TORA	reach-tanh	✓	12.420
TORA	reach-sigmoid	✓	8.980
Single Pendulum	reach	✓	3.650
Double Pendulum	less-robust	?	5.760
Double Pendulum	more-robust	✗	4.360
Airplane	continuous	✗	10.690
VCAS	middle-19.5	✓	0.210
VCAS	middle-22.5	✓	0.100
VCAS	middle-25.5	✗	0.040
VCAS	middle-28.5	✗	0.070
VCAS	worst-19.5	✓	0.130
VCAS	worst-22.5	✗	0.040
VCAS	worst-25.5	✗	0.040
VCAS	worst-28.5	✗	0.040
Attitude Control	avoid	✓	6.520
QUAD	reach	✓	214.890
Docking	constraint	?	79.570
NAV	standard	?	8.440
NAV	robust	✓	13.930
CartPole*	reach	✓	26.110

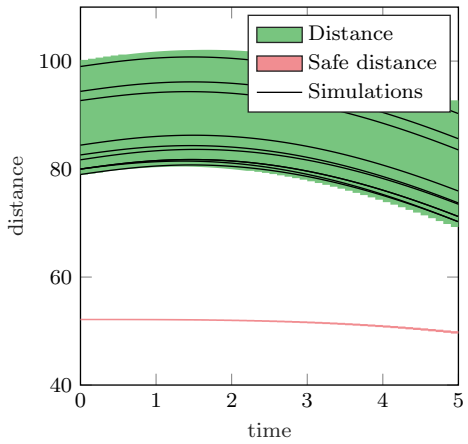


Figure 5: ***CORA.*** Computed reachable set and simulations of the ACC benchmark.

4.1.1 ACC

CORA is able to verify this benchmark. The computed reachable set as well as some simulations are visible in Fig. 5.

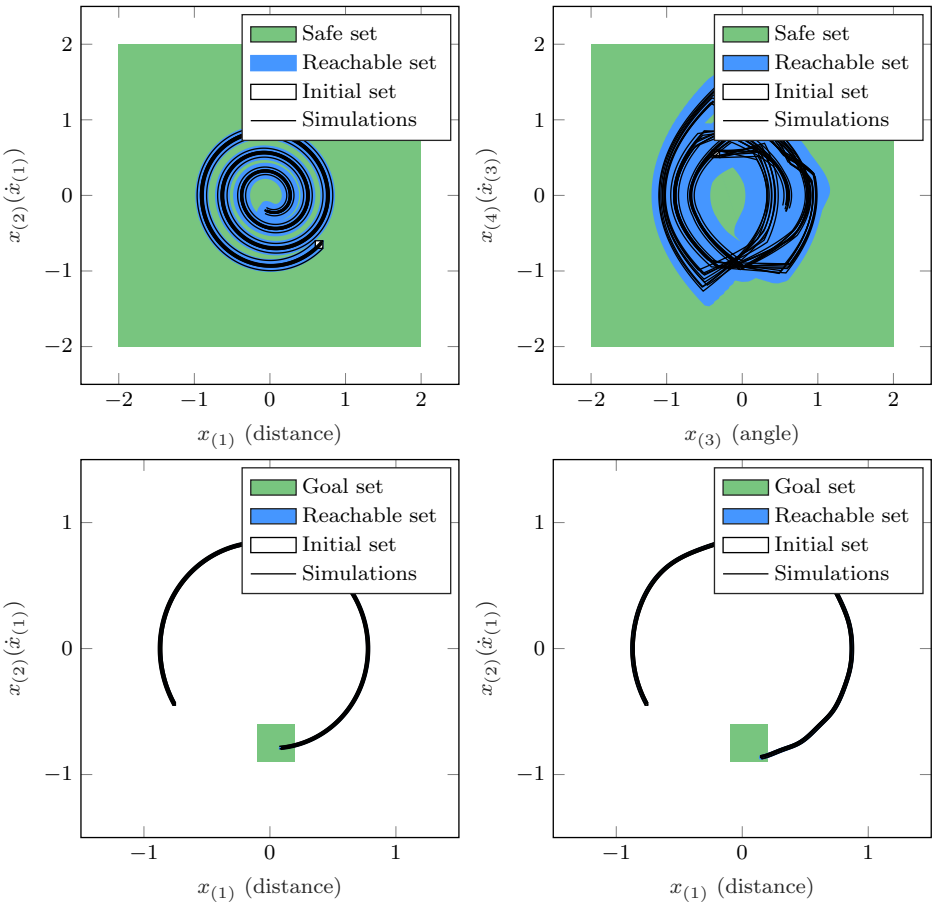


Figure 6: ***CORA.*** Computed reachable set and simulations of the TORA benchmark: ReLU contoller (top), tanh controller (bottom left), and sigmoid controller (bottom right).

4.1.2 TORA

CORA is able to verify both the remain and the reach instance for all controllers of this benchmark. The computed reachable set as well as some simulations are visible in Fig. 6.

4.1.3 Unicycle

CORA is able to verify this benchmark. The computed reachable set as well as some simulations are visible in Fig. 7.

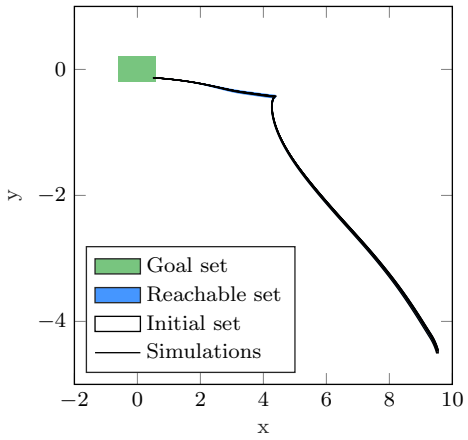


Figure 7: ***CORA.*** Computed reachable set and simulations of the unicycle benchmark.

4.1.4 VCAS

The VCAS benchmark has discrete time steps and multiple controllers, which is currently not supported by CORA. Thus, a custom algorithm was built for this benchmark. To deal with the discrete input set $\dot{h}_0(0) \in \{-19.5, -22.5, -25.5, -28.5\}$, we run the algorithm with each element of the input set individually. As proposed in the benchmark specifications, we show the results when always the middle acceleration of the controllers is chosen and the results when always the worst acceleration is chosen.

VCAS (middle acceleration) Here we always use the middle of the possible accelerations. We are able to verify the benchmark for $\dot{h}_0(0) \in \{-19.5, -22.5\}$ and can show violations for $\dot{h}_0(0) \in \{-25.5, -28.5\}$. The computed reachable set along with some simulations are shown in Figure 8.

VCAS (worst acceleration) Here we always use the worst possible acceleration. We are able to verify the benchmark for $\dot{h}_0(0) \in \{-19.5\}$ and can show violations for $\dot{h}_0(0) \in \{-22.5, -25.5, -28.5\}$. The computed reachable set along with some simulations are shown in Figure 9.

4.1.5 Single Pendulum

CORA is able to verify this benchmark. The computed reachable set as well as some simulations are visible in Fig. 10.

4.1.6 Double Pendulum

CORA is able to falsify the instance using the more robust controller by providing a verified simulation run going outside the safe set, whereas the result for the less robust controllers remains unknown. The verified simulation run violating the specification is visible in Fig. 11.

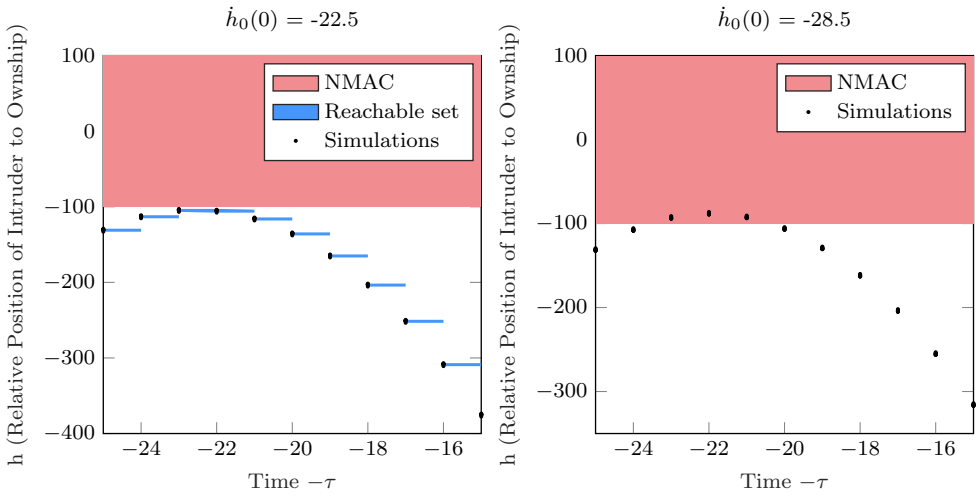


Figure 8: ***CORA.*** Computed reachable set of the VCAS benchmark with middle acceleration.

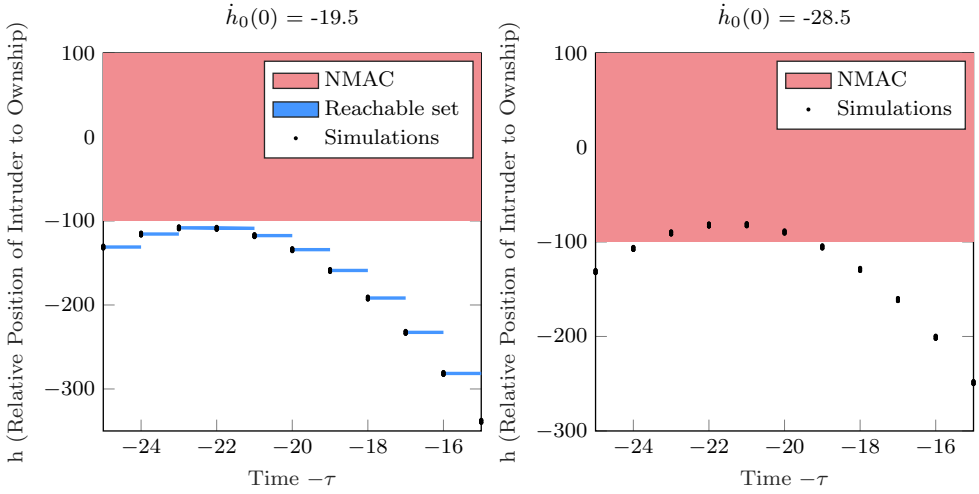


Figure 9: ***CORA.*** Computed reachable set of the VCAS benchmark with worst acceleration.

4.1.7 Airplane

CORA is able to falsify this benchmark by providing a verified simulation run going outside the safe set. The verified simulation run violating the specification is visible in Fig. 12.

4.1.8 Attitude Control

CORA is able to verify this benchmark. The computed reachable set as well as some simulations are visible in Fig. 13.

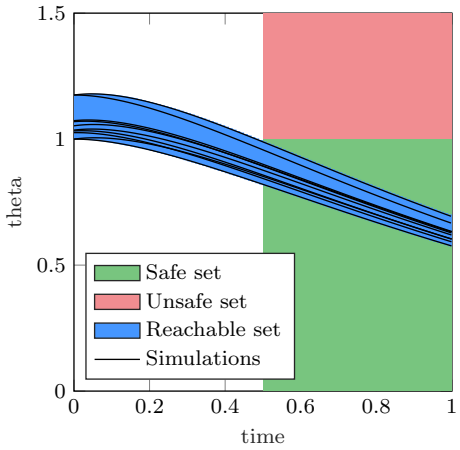


Figure 10: ***CORA.*** Computed reachable set and simulations of the single pendulum benchmark.

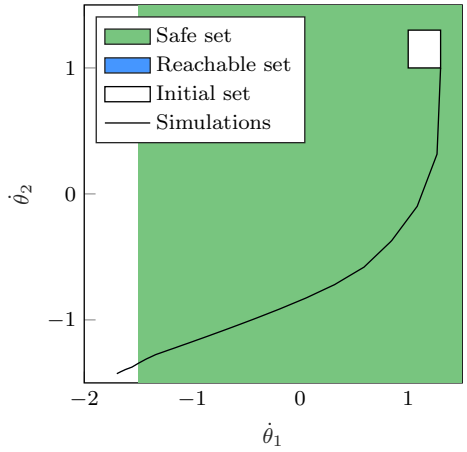


Figure 11: ***CORA.*** Verified simulation run using the more robust controller violating the specification of the double pendulum benchmark.

4.1.9 Quadrotor

CORA is able to verify this benchmark. The computed reachable set as well as some simulations are visible in Fig. 14.

4.1.10 2D Spacecraft Docking

The spacecraft docking benchmark appeared difficult to verify for our approach. While the simulations seem to be stable, the reachable set explodes over time and thus we are unable to verify the benchmark. The computed reachable set along with some simulations are shown in Figure 15.

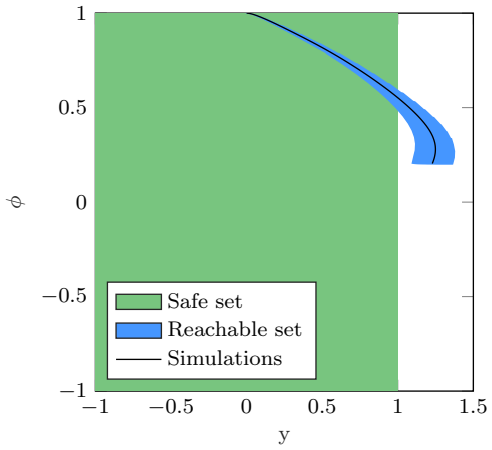


Figure 12: ***CORA.*** Verified simulation run violating the specification of the airplane benchmark.

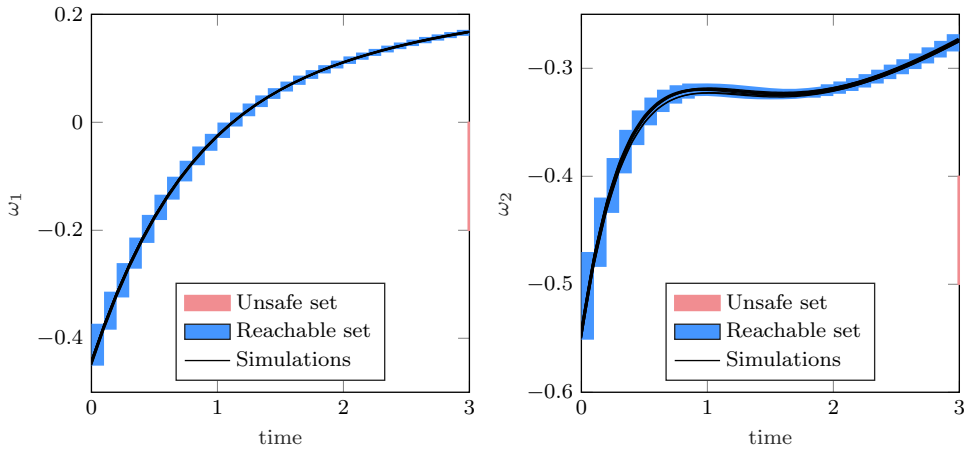


Figure 13: ***CORA.*** Computed reachable set and simulations of the attitude control benchmark.

4.1.11 Navigation Task

CORA is able to verify the instance with the controller obtained via set-based adversarial training, while the standard controller could not be verified although no violating simulation run was found. The computed reachable set along with some simulations are shown in Figure 16.

4.1.12 CartPole

CORA is able to verify this benchmark for a smaller initial set. The computed reachable set along with some simulations are shown in Figure 17.

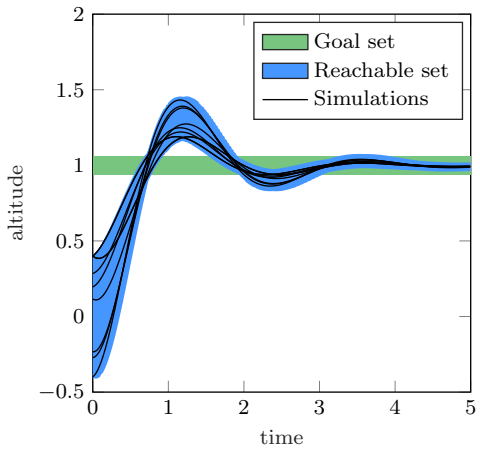


Figure 14: ***CORA.*** Computed reachable set and simulations of the quadrotor benchmark.

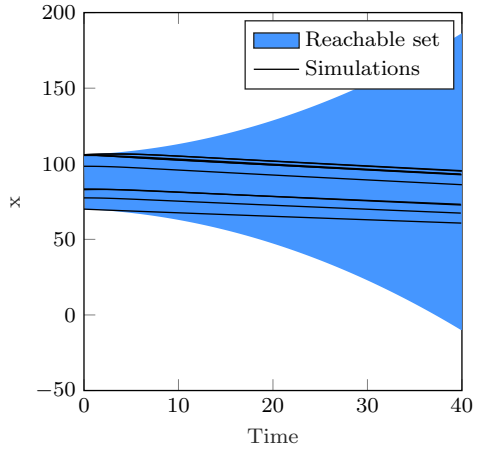


Figure 15: ***CORA.*** Computed reachable set and simulations of the 2D spacecraft docking benchmark.

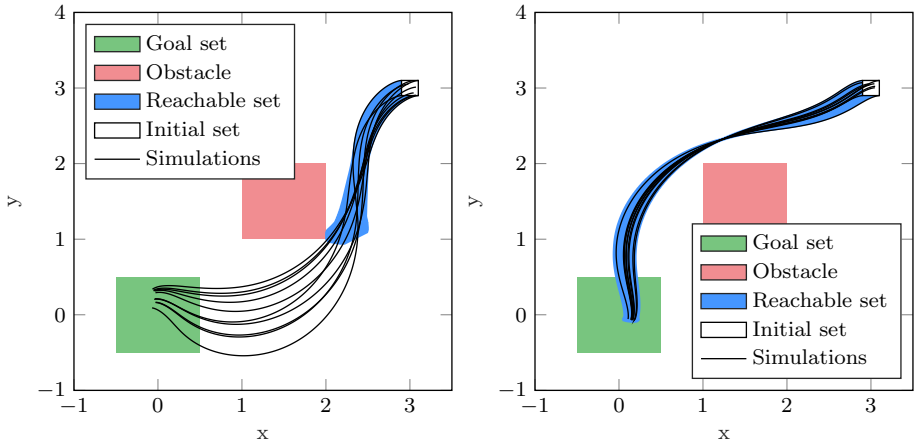


Figure 16: ***CORA.*** Computed reachable set and simulations of the navigation task benchmark: Standard controller (left) and robust contoller (right).

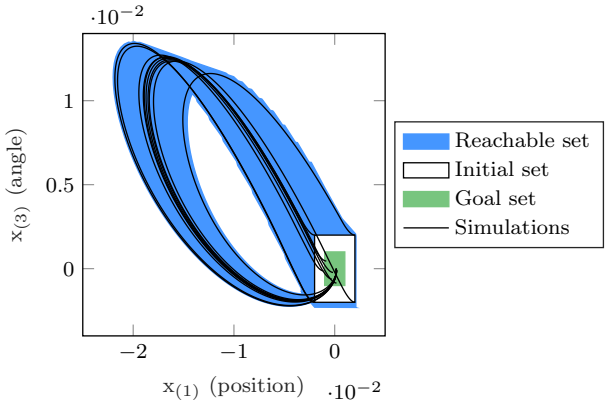


Figure 17: ***CORA.*** Computed reachable set and simulations of the cartpole benchmark.

4.2 CROWN-Reach

This subsection presents the results of CROWN-Reach. Here we briefly introduce the specific setting for each benchmark. All the implementation and parameters can be found in the repeatability repository. An overview of the results is given in Tab. 2.

4.2.1 ACC

CROWN-Reach is able to verified this benchmark. The reachable sets of both "distance" D_{rel} and "safe distance" D_{safe} are shown in Fig. 18.

4.2.2 TORA

CROWN-Reach is able to verify all three instances of this benchmark. The computed reachable sets of the system are shown in Fig. 19. Note that for the instance under the ReLU controller (with specification 1), the initial set of x_1 and x_2 is uniformly divided into 4×3 subsets.

4.2.3 Unicycle

CROWN-Reach is able to verify this benchmark. The reachable sets are shown in Fig. 20.

Table 2: ***CROWN-Reach.*** Overview of results: Verified (✓), falsified (✗), and unknown (**?**).

Benchmark	**Instance**	**Result**	**Time [s]**
ACC	safe-distance	✓	2.590
Airplane	continues	✗	6.580
Attitude Control	avoid	✓	3.320
Single Pendulum	reach	✓	0.630
TORA	remain	✓	2.750
TORA	reach-sigmoid	✓	6.230
TORA	reach-tanh	✓	5.810
Unicycle	reach	✓	9.650
CartPole	reach	?	6.210
VCAS	worst-19.5	✓	0.270
VCAS	worst-22.5	✗	0.040
VCAS	worst-25.5	✗	0.030
VCAS	worst-28.5	✗	0.030
VCAS	middle-19.5	✓	0.270
VCAS	middle-22.5	✓	0.270
VCAS	middle-25.5	✗	0.030
VCAS	middle-28.5	✗	0.030
Double Pendulum	more-robust	✗	1.170
Double Pendulum	less-robust	✓	68.480
NAV	robust	✓	20.650
NAV	standard	✓	130.120
Quadrotor	reach	✓	3465.170

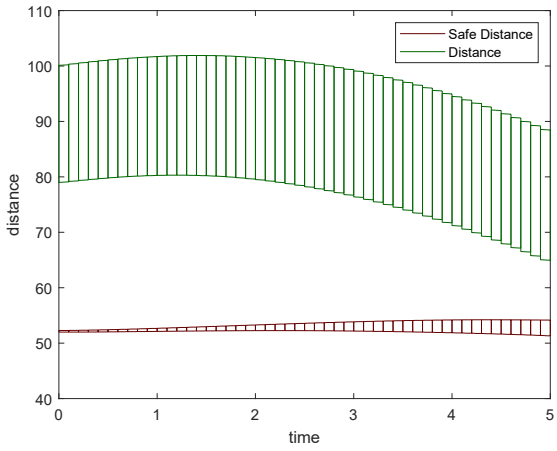


Figure 18: ***CROWN-Reach.*** Computed reachable sets of distance and safe distance in the following 5 seconds.

4.2.4 VerticalCAS

The VCAS benchmark has multiple controllers. To handle this benchmark, we treat the four possible inputs $\dot{h}_0(0) \in \{-19.5, -22.5, -25.5, -28.5\}$ separately and fix the acceleration selection strategies between "always choosing the middle one" and "always choosing the worst one". At each control step, CROWN-Reach bounds the outputs of the neural network controller and verifies that the advisory option is unique. To conclude, CROWN-Reach is able to show that when using the "worst" strategy, instances with $\dot{h}_0(0) = -19.5$ is verified. When using the "middle" strategy, instances with $\dot{h}_0(0) = -19.5$ or $\dot{h}_0(0) = -22.5$ are verified. All other instances are falsified by random simulation. The result visualizations are shown in Fig. 21.

4.2.5 Single Pendulum

CROWN-Reach is able to verify this benchmark. The reachable sets are shown in Fig. 22.

4.2.6 Double Pendulum

CROWN-Reach is able to falsify the instance under the more robust controller by running a simulation starting from the initial point $(1.3, 1.3, 1.3, 1.3)$, and is able to verify the instance under the less robust controller by uniformly dividing the initial set into $5 \times 5 \times 3 \times 3$ subsets. The result visualizations are shown in Fig. 23.

4.2.7 Airplane

CROWN-Reach is able to falsify this benchmark by showing that the trajectory starting from a counterexample initial state reach out of the safe set, as shown in Fig. 24.

4.2.8 Attitude Control

CROWN-Reach is able to verify this benchmark. The computed reachable sets are shown in Fig. 25.

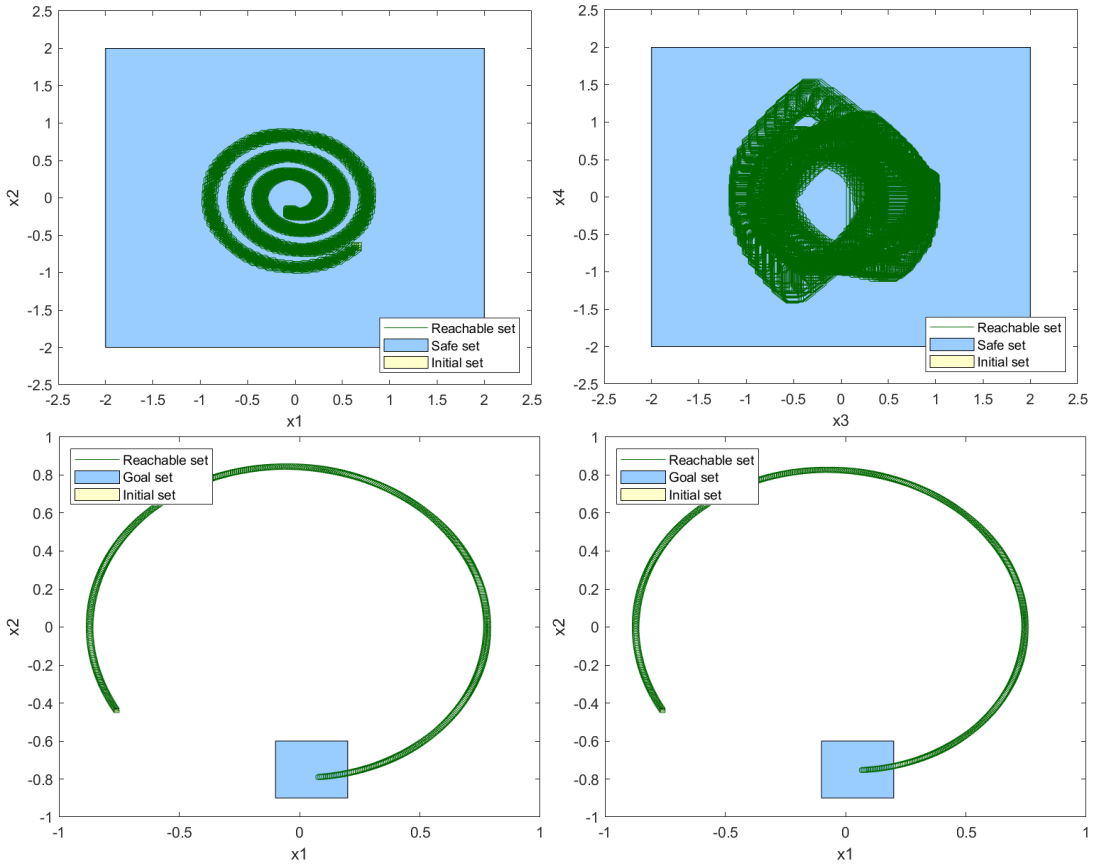


Figure 19: ***CROWN-Reach.*** Computed reachable sets of the TORA benchmark under the ReLU controller (top), the ReLU/tanh controller (bottom left), and the sigmoid controller (bottom right). For the instance with sigmoid controller, our result is based on the specification used in the competition, where the output of the neural network $f(\mathbf{x})$ is post-processed as $u = 22 \cdot (f(\mathbf{x}) - 0.5)$ instead of $u = 11 \cdot f(\mathbf{x})$ described in Sec. 3.2. All three instances are verified.

4.2.9 Quadrotor

CROWN-Reach is able to verify this benchmark with input splits. The initial set is uniformly divided into $8 \times 8 \times 8 \times 2 \times 1 \times 1$ subsets and verified in parallel. The reachable sets are shown in Fig. 26.

4.2.10 Navigation Task

CROWN-Reach is able to verify both instances of this benchmark with input splits. For the standard instance, the initial set is uniformly divided into $40 \times 16 \times 1 \times 1$ subsets and verified in parallel. For the robust instance, the initial set is uniformly divided into $5 \times 5 \times 1 \times 1$ subsets and verified in parallel. The reachable sets are shown in Fig. 27.

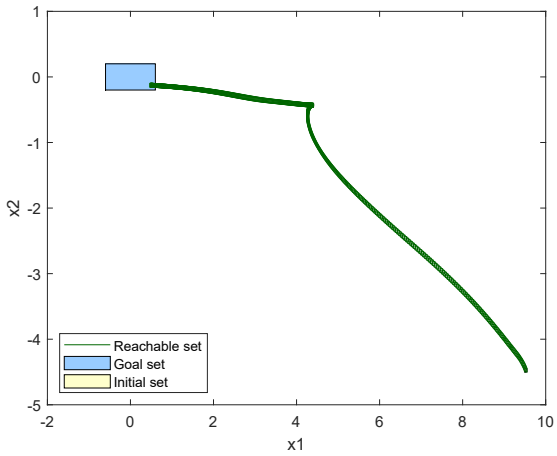


Figure 20: ***CROWN-Reach.*** Computed reachable sets of the Unicycle benchmark. The target set is proved to be reachable.

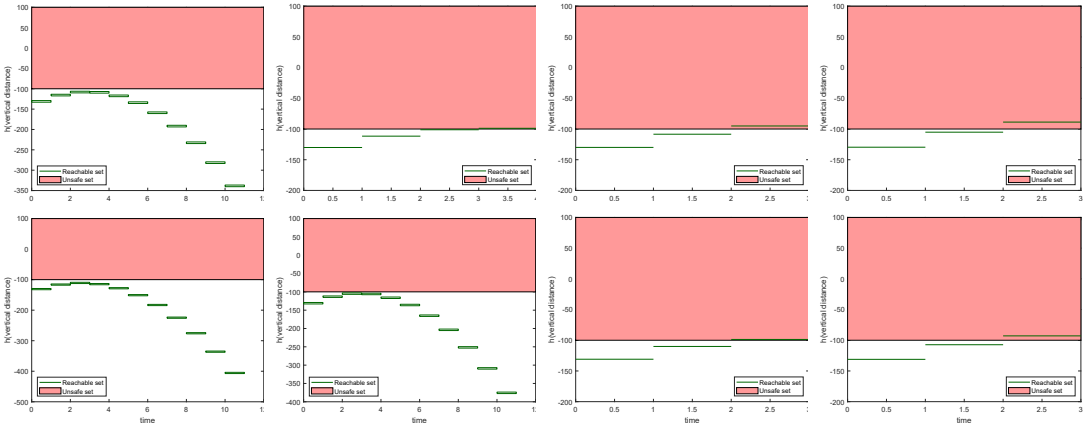


Figure 21: ***CROWN-Reach.*** Computed reachable sets (for verified instances) and simulation trajectories (for falsified instances) of the VCAS benchmark. The strategy for choosing acceleration is fixed to either "worst" (top) or "middle" (bottom). The initial value of $\dot{h}_0$ is chosen from $\{-19.5, -22.5, -25.5, -28.5\}$ (from left to right).

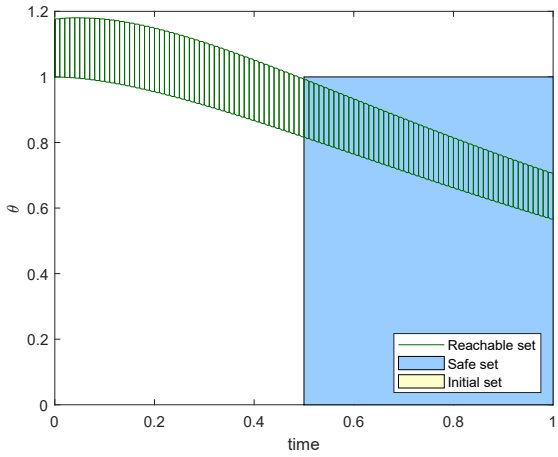


Figure 22: ***CROWN-Reach.*** Computed reachable sets of the Single Pendulum benchmark.

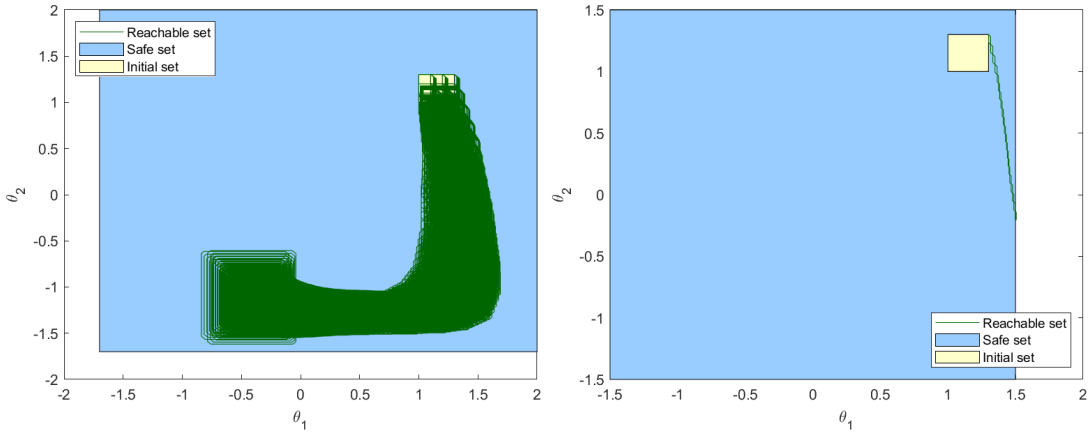


Figure 23: ***CROWN-Reach.*** Result visualizations for the benchmark Double Pendulum, including the instance under the less robust controller (left) and the one under the more robust controller (right). For the falsified instance, we plot one counterexample trajectory. For the verified one, we show its reachable sets.

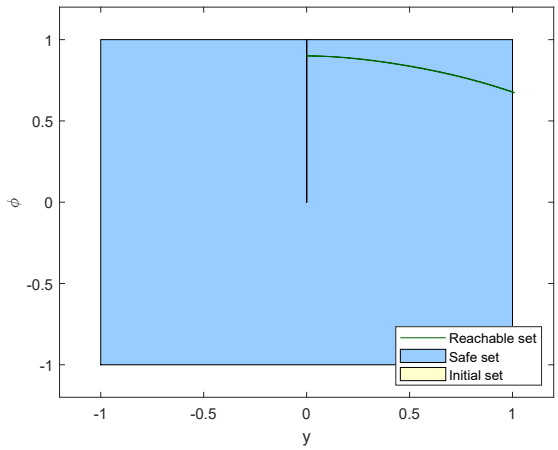


Figure 24: ***CROWN-Reach.*** A trajectory from an initial point reaches out of the safe set, falsifying the Airplane benchmark.

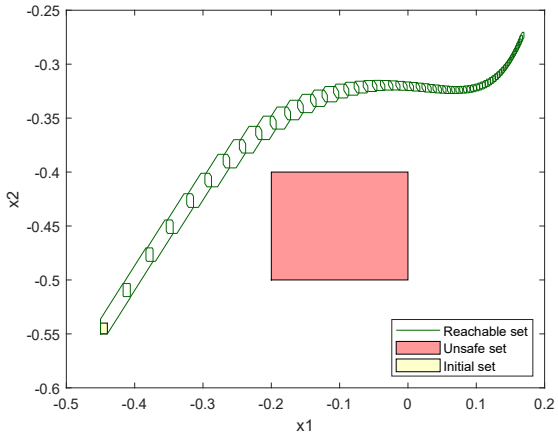


Figure 25: ***CROWN-Reach.*** Computed reachable sets of the Attitude Control benchmark.

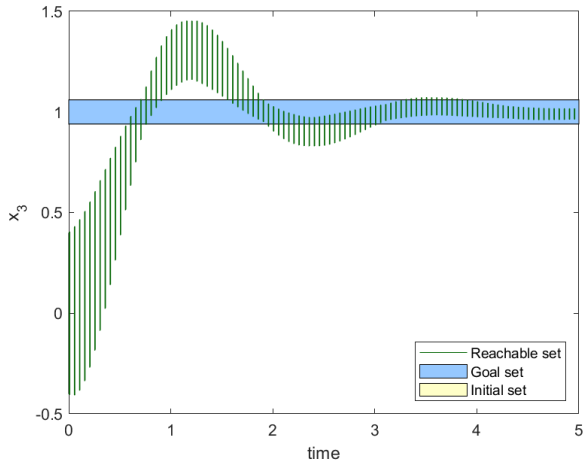


Figure 26: ***CROWN-Reach.*** Computed reachable sets of the Quadrotor benchmark. Here for visualization, reachable sets are sampled on the dimension *time*.

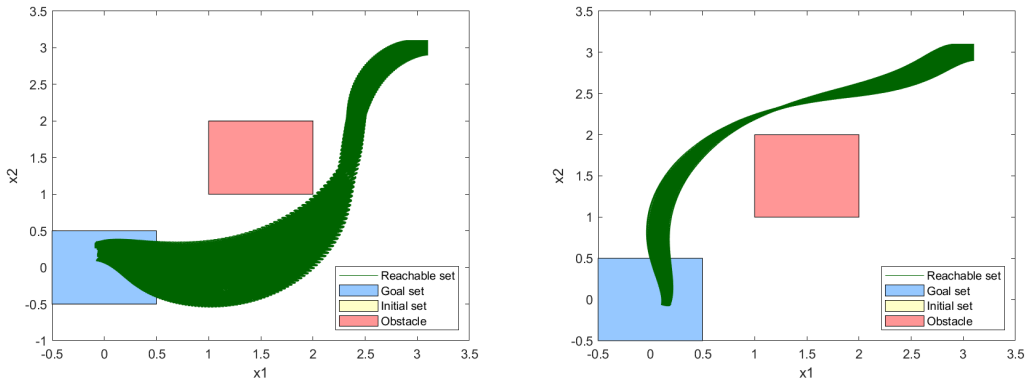


Figure 27: ***CROWN-Reach.*** Computed reachable sets of the Navigation Task benchmark. Both the standard instance (left) and the robust one (right) are verified.

4.3 GoTube v2.0

We present the results for all benchmarks utilizing GoTube v2.0. We were able to specify all benchmarks except Spacecraft Docking 2D, were the specification could only be done on an input set with half the radius of the original one. The verification for all benchmarks were done with a confidence level of 90% ($\gamma = 0.1$). For all other details and parameters used in the reachability and validation of all benchmarks (e.g. tightness factor μ) we refer to the submission package code available at https://gitlab.com/goranf/ARCH-COMP/-/tree/master/2024/AINNCS/GoTube. An overview of the results is given in Tab. 3.

4.3.1 ACC

GoTube is able to verify the specifications automatically. Figure 28 shows the reachable values of the safety function $f = D_{\text{safe}} - D_{\text{rel}}$ plotted over time (where $f \geq 0$ indicates a safe behaviour as described in the benchmarks specifications).

Table 3: ***GoTube v2.0.*** Overview of results: Verified (✓), falsified (✗), and unknown (**?**).

Benchmark	**Instance**	**Result**	**Time** [s]
ACC		✓	90.450
Attitude-Control		✓	10.820
QUAD		✓	7.900
Airplane		✗	1.940
Docking	full_initial_set_trajectories_only	✓	4.100
Docking	half_initial_set_with_safety_property	✓	54.220
Double_Pendulum	less_robust_controller	✓	5.650
Double_Pendulum	more_robust_controller	✗ (tube)	1.170
Single_Pendulum		✓	2.170
Benchmark9-Tora		✓	4.980
Tora_Heterogeneous	sigmoid_controller	✓	1.410
Tora_Heterogeneous	relu_tanh_controller	✓	2.580
Benchmark9-Unicycle		✓	3.530
VCAS	mid_acceleration_h0_19p5	✓	3.010
VCAS	mid_acceleration_h0_22p5	✓	3.070
VCAS	mid_acceleration_h0_25p5	✗	3.560
VCAS	mid_acceleration_h0_28p5	✗	2.220
VCAS	worst_acceleration_h0_19p5	✓	3.200
VCAS	worst_acceleration_h0_22p5	✗	3.820
VCAS	worst_acceleration_h0_25p5	✗	2.230
VCAS	worst_acceleration_h0_28p5	✗	2.260
CartPole		✓	39.870
NAV	less_robust_controller	✓	5.020
NAV	more_robust_controller	✓	4.990

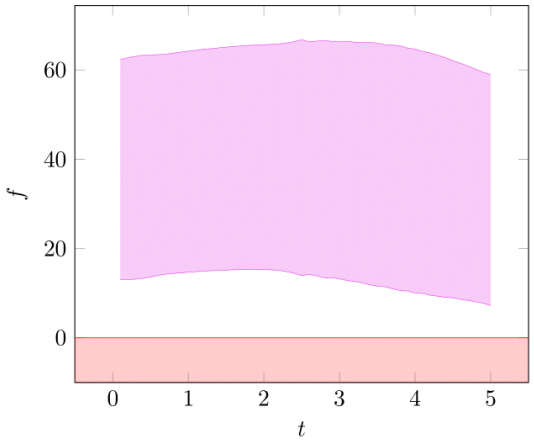


Figure 28: ***GoTube.*** Analysis results for the ACC benchmark with reachable values of the safety function f over time. ≥ 0 verifies the controller.

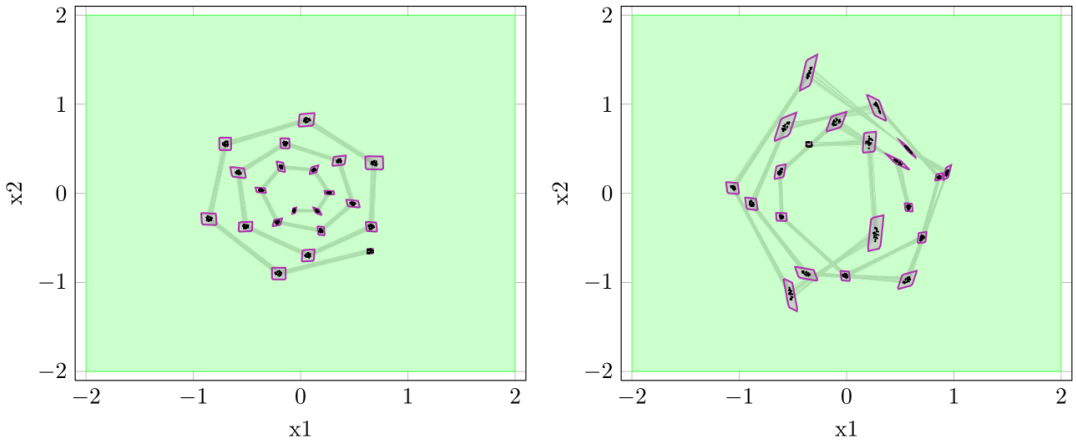


Figure 29: ***GoTube.*** Reachable set (calculated only at the given time steps) and specifications for the TORA benchmark.

4.3.2 TORA

All specifications can be verified by GoTube automatically. The reachable set, the specifications and 25 simulations can be seen in Figure 29 for the ReLU controller with specification 1 as well as in Figure 30 for both the sigmoid and tanh controllers for specification 2. The simulations were only calculated correctly for the given time steps. Their connections (plotted in gray) are only linear interpolations. Also the reachable set is only calculated for the given time steps.

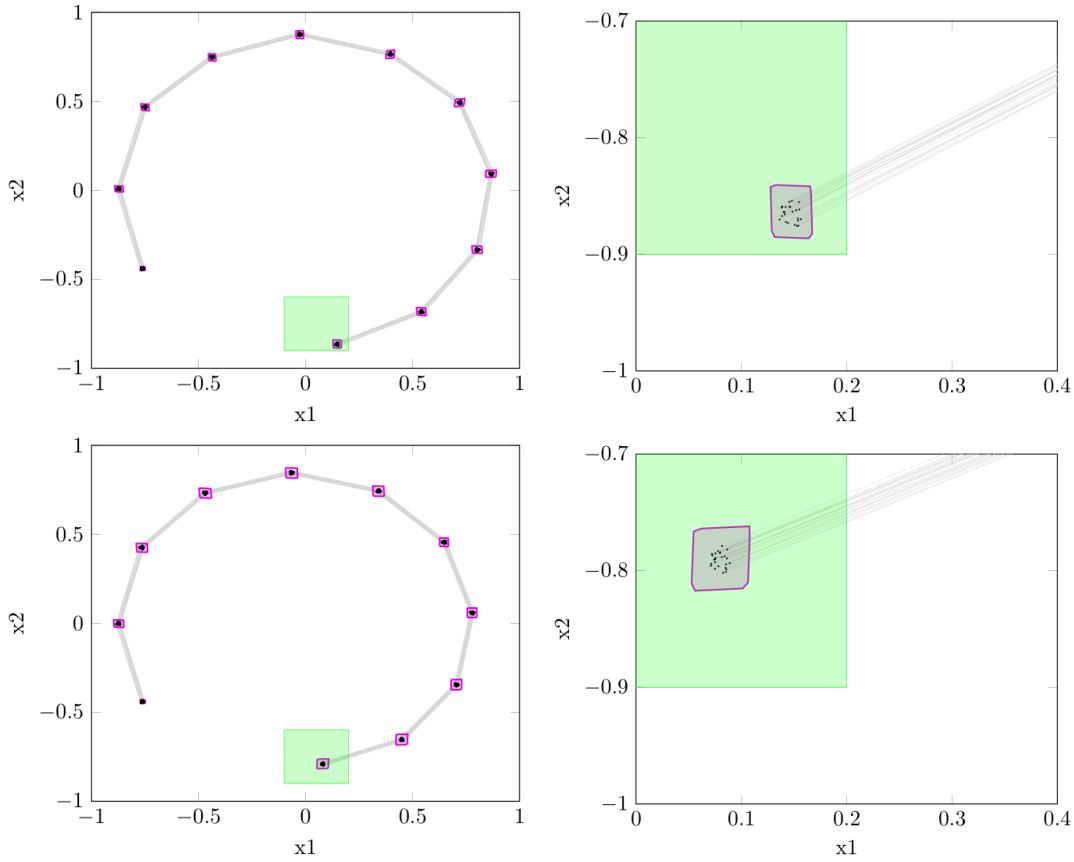


Figure 30: ***GoTube.*** Reachable set (calculated only at the given time steps) and specifications for the TORA (Heterogeneous) benchmark. Sigmoid controller (top) and Tanh controller (bottom) as well as plots zoomed in on the goal set (right).

4.3.3 Unicycle

GoTube is able to fully verify the specifications automatically. Figure 31 illustrates the reachable set, the specifications as well as 25 simulations. The simulations were only calculated correctly for the given time steps. Their connections (plotted in gray) are only linear interpolations. Also the reachable set is only calculated for the given time steps.

4.3.4 VCAS

For GoTube to handle the discrete initial set of $\dot{h}_0(0)$ the analysis was run separately for every starting value.

VCAS (middle acceleration) The middle of all possible accelerations is used. GoTube is able to verify the specification of this benchmark for $\dot{h}_0(0) \in \{-19.5, -22.5\}$ and falsify them for $\dot{h}_0(0) \in \{-25.5, -28.5\}$. Figure 32 illustrates the reachable sets, the specifications as well as 25 simulations for each starting value $\dot{h}_0(0)$.

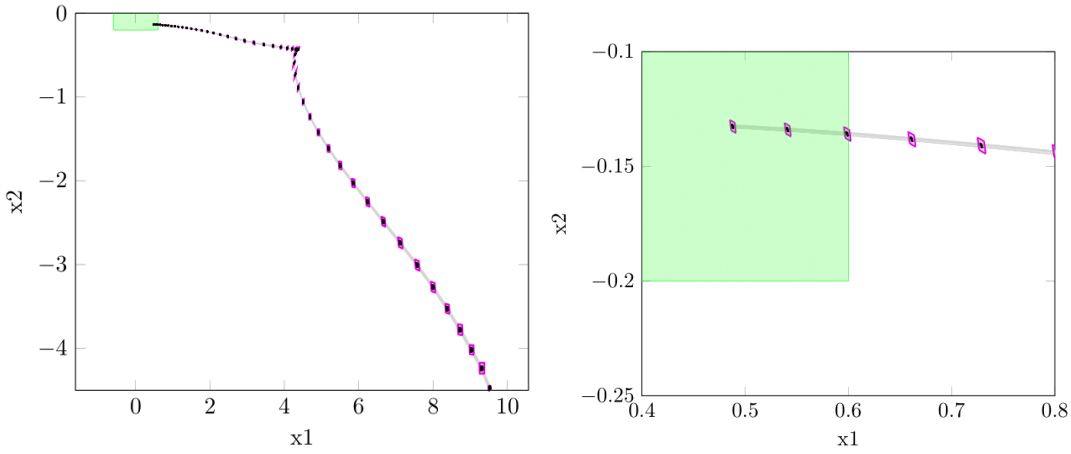


Figure 31: ***GoTube.*** Reachable set and specifications for the Unicycle benchmark.

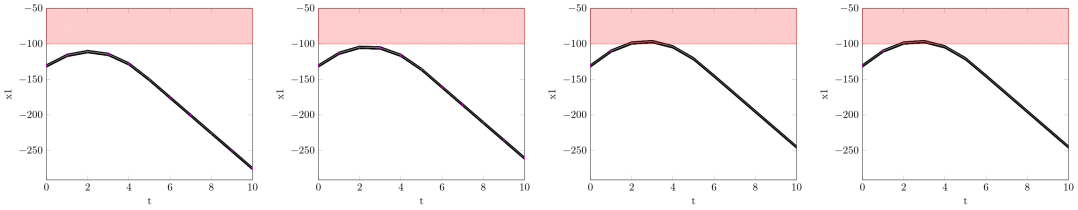


Figure 32: ***GoTube.*** Reachable set and specifications for the VCAS benchmark with the controller selecting the middle acceleration for $\dot{h}_0(0) = 19.5$, $\dot{h}_0(0) = 22.5$, $\dot{h}_0(0) = 25.5$ and $\dot{h}_0(0) = 28.5$ (from left to right).

VCAS (worst acceleration) The worst of all possible accelerations is used. GoTube is able to verify the specification of this benchmark for $\dot{h}_0(0) = -19.5$ and falsify them for $\dot{h}_0(0) \in \{-22.5, -25.5, -28.5\}$. Figure 33 illustrates the reachable sets, the specifications as well as 25 simulations for each starting value $\dot{h}_0(0)$.

4.3.5 Single Pendulum

GoTube is able to fully verify the specifications automatically. Figure 34 illustrates the reachable set, the specifications as well as 25 simulations.

4.3.6 Double Pendulum

GoTube is able to fully verify the specifications for the less robust controller and falsify the specifications of the more robust controller automatically. Figure 35 illustrates the reachable set, the specifications as well as 25 simulations each. The simulations were only calculated correctly for the given time steps. Their connections (plotted in gray) are only linear interpolations. Also the reachable set is only calculated for the given time steps.

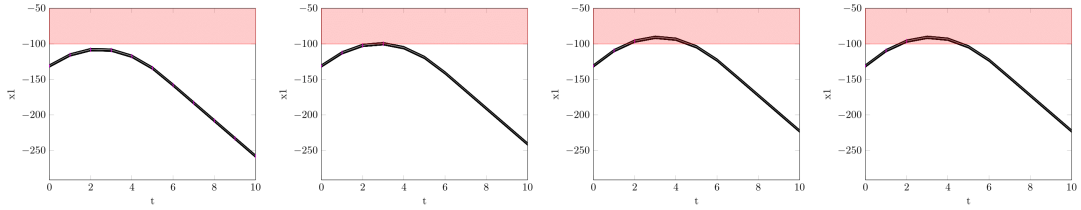


Figure 33: ***GoTube.*** Reachable set and specifications for the VCAS benchmark with the controller selecting the middle acceleration for $\dot{h}_0(0) = 19.5$, $\dot{h}_0(0) = 22.5$, $\dot{h}_0(0) = 25.5$ and $\dot{h}_0(0) = 28.5$ (from left to right).

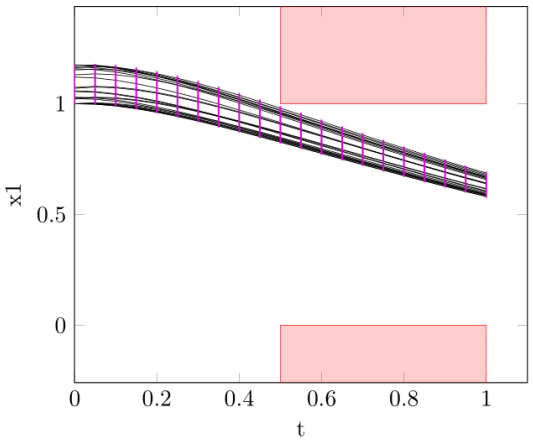


Figure 34: ***GoTube.*** Reachable set (calculated only at the given time steps) and specifications for the Single Pendulum benchmark.

4.3.7 Airplane

GoTube is able to falsify the specifications automatically. Figure 36 illustrates the reachable set, the specifications as well as 25 simulations. The reachable set is only plotted until the first violation of the specifications.

4.3.8 Attitude Control

GoTube is able to fully verify the specifications automatically. Figure 37 illustrates the reachable set, the specifications as well as 25 simulations. The simulations were only calculated correctly for the given time steps. Their connections (plotted in gray) are only linear interpolations. Also the reachable set is only calculated for the given time steps.

4.3.9 Quadrotor

GoTube is able to fully verify the specifications automatically. Figure 38 illustrates the reachable set, the specifications as well as 25 simulations.

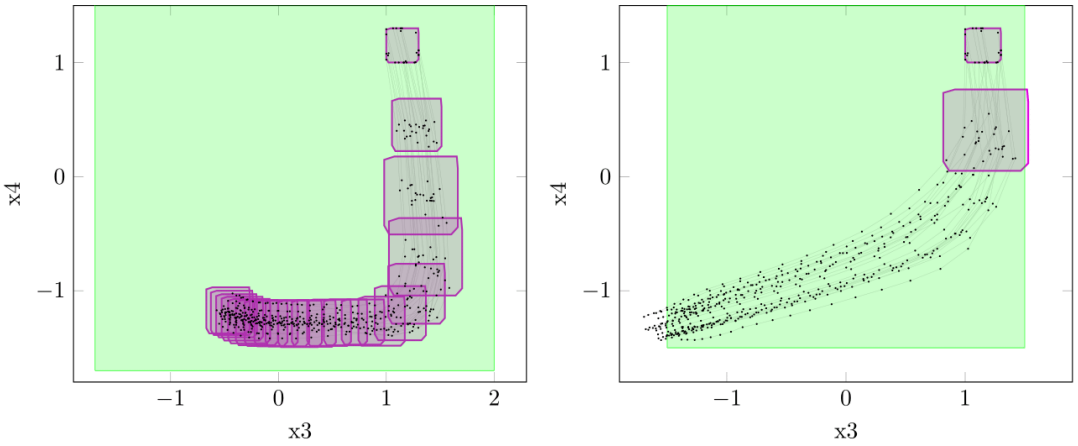


Figure 35: ***GoTube.*** Reachable set (calculated only at the given time steps) and specifications for the Double Pendulum benchmark with the less robust (left) and the more robust (right) controllers.

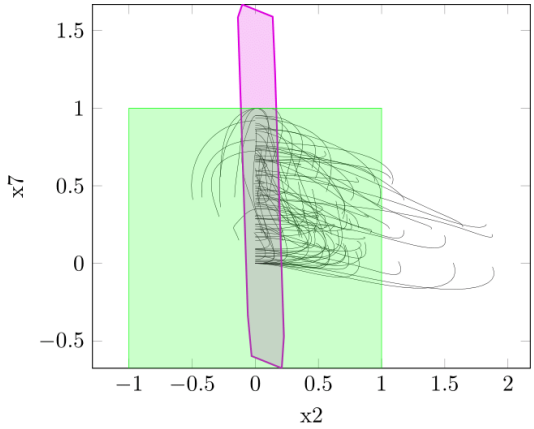


Figure 36: ***GoTube.*** Reachable set and specifications for the Airplane benchmark. The reachable set is only constructed for the first time step, as the set already violates the specification.

4.3.10 2D Spacecraft Docking

GoTube was not able to verify this benchmark for the full initial set. It was however possible to construct the reachable set at every time step by using a larger tightness factor. The benchmark could be verified automatically for an initial set with half the radius of the original initial set. The reachable set of the original initial set is plotted in Figure 39.

4.3.11 Navigation Task

Both controllers could automatically be verified by GoTube. Figure 40 illustrates the reachable set, the specifications as well as 25 simulations each. The simulations were only calculated correctly for the given time steps. Their connections (plotted in gray) are only linear interpolations. Also the reachable set is only calculated for the given time steps.

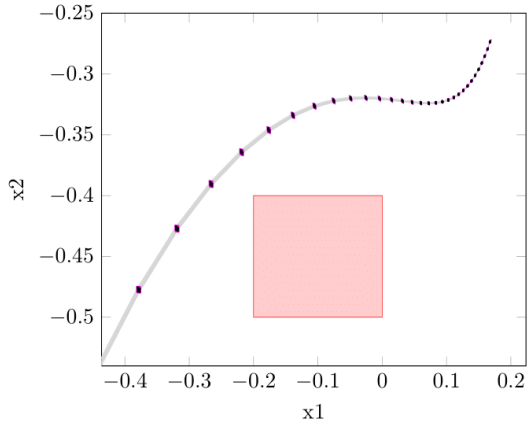


Figure 37: ***GoTube.*** Reachable set (calculated only at the given time steps) and specifications for the Attitude Control benchmark.

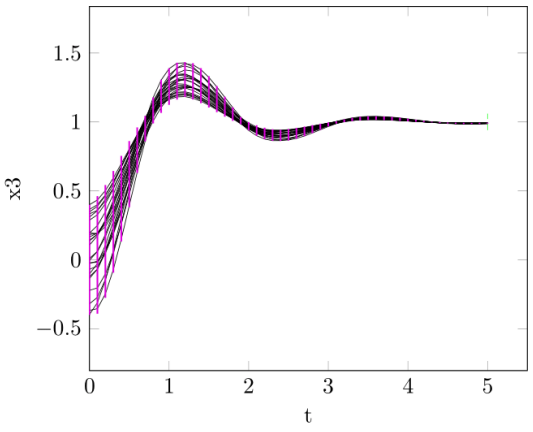


Figure 38: ***GoTube.*** Reachable set (calculated only at the given time steps) and specifications for the Quadrotor benchmark.

4.3.12 CartPole

GoTube is able to fully verify the specifications automatically. Figure 41 illustrates the reachable set, the specifications as well as 25 simulations.

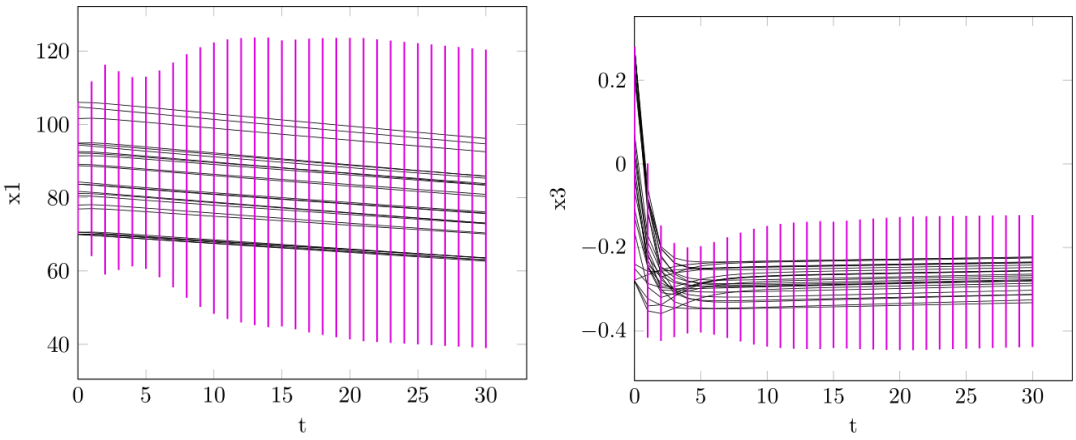


Figure 39: ***GoTube.*** Reachable set for the Docking benchmark using the whole initial set.

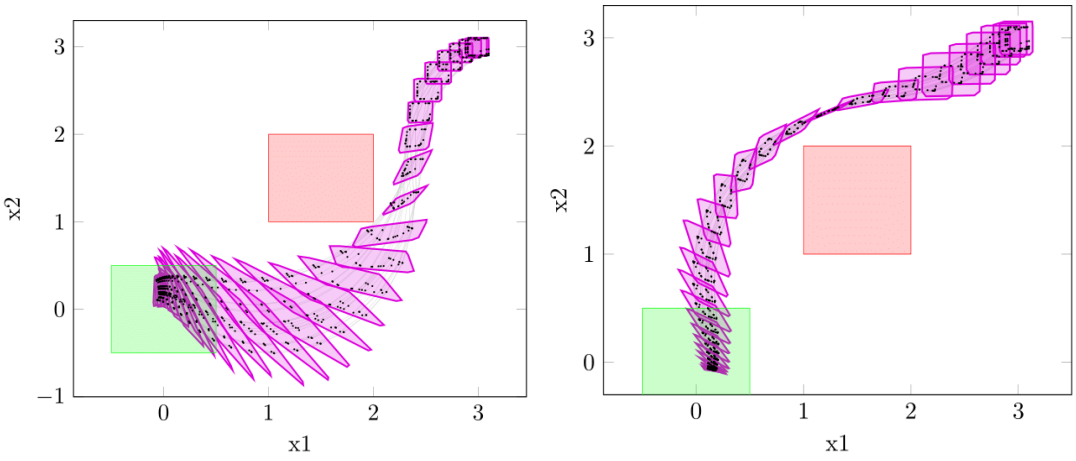


Figure 40: ***GoTube.*** Reachable set (calculated only at the given time steps) and specifications for the Navigation benchmark with the less robust (left) and the more robust (right) controllers.

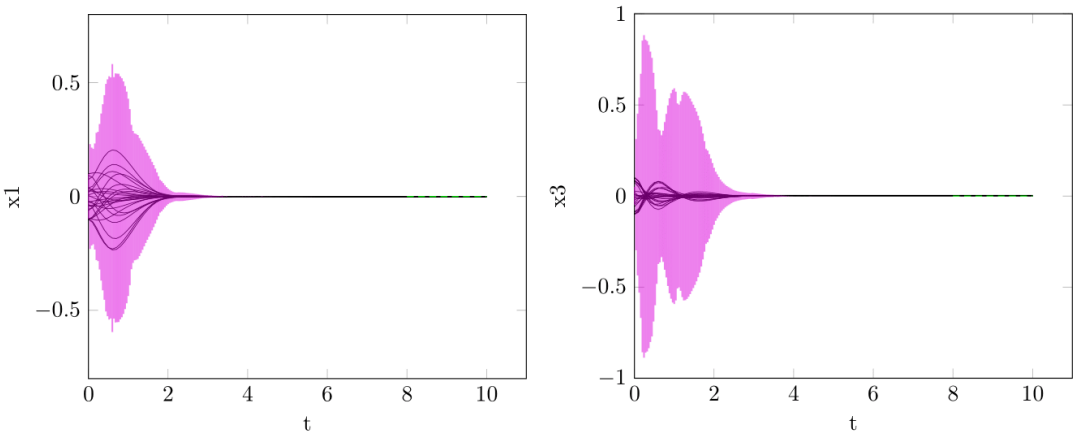


Figure 41: ***GoTube.*** Reachable set for the Cartpole benchmark.

4.4 JuliaReach

This subsection presents the results of *JuliaReach*. For each problem, JuliaReach uses slightly different settings as described below. An overview of the results is given in Tab. 4.

4.4.1 ACC

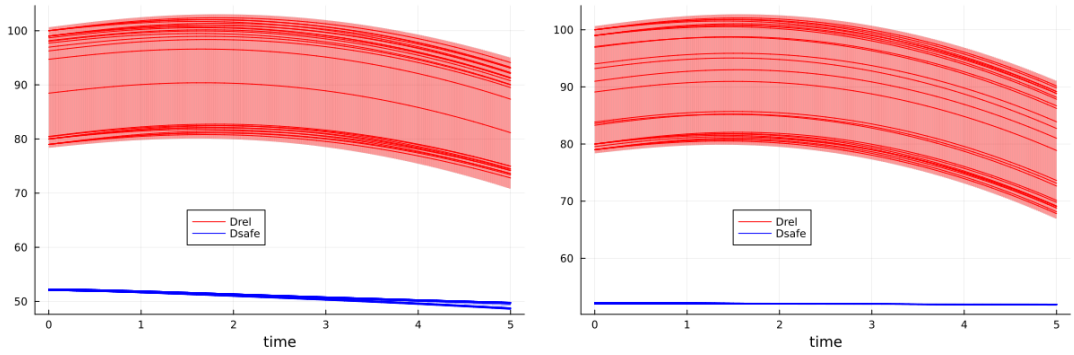


Figure 42: ***JuliaReach.*** Analysis results for the ACC benchmark under the ReLU controller (left) resp. the tanh controller (right). The plots additionally show simulations.

Using the parameters `abstol=1e-3, orderT=5, orderQ=1`, JuliaReach verifies the specification in 0.8 and 0.4 s, respectively. Figure 42 shows the reach sets of D_{rel} and D_{safe}.

Table 4: ***JuliaReach.*** Overview of results: Verified (✓), falsified (✗), and unknown (?).

Benchmark	**Instance**	**Result**	**Time [s]**
ACC	ReLU	✓	0.800
ACC	tanh	✓	0.420
TORA	ReLU	✓	184.330
TORA	ReLUtanh	✓	0.240
TORA	sigmoid	✓	1.320
Unicycle	constant w	✓	36.720
VCAS central acceleration	h'(0) = -19.5	✓	0.010
VCAS central acceleration	h'(0) = -22.5	✓	0.000
VCAS central acceleration	h'(0) = -25.5	✗	0.000
VCAS central acceleration	h'(0) = -28.5	✗	0.000
SinglePendulum		✓	14.040
DoublePendulum	less robust	✓	1520.120
DoublePendulum	more robust	✗	0.980
Airplane		✗	2.440
AttitudeControl		✓	4.450
Quadrotor	small X0	✓	10.410
Spacecraft	small X0	✓	4.030
NAV	robust	✓	4.490

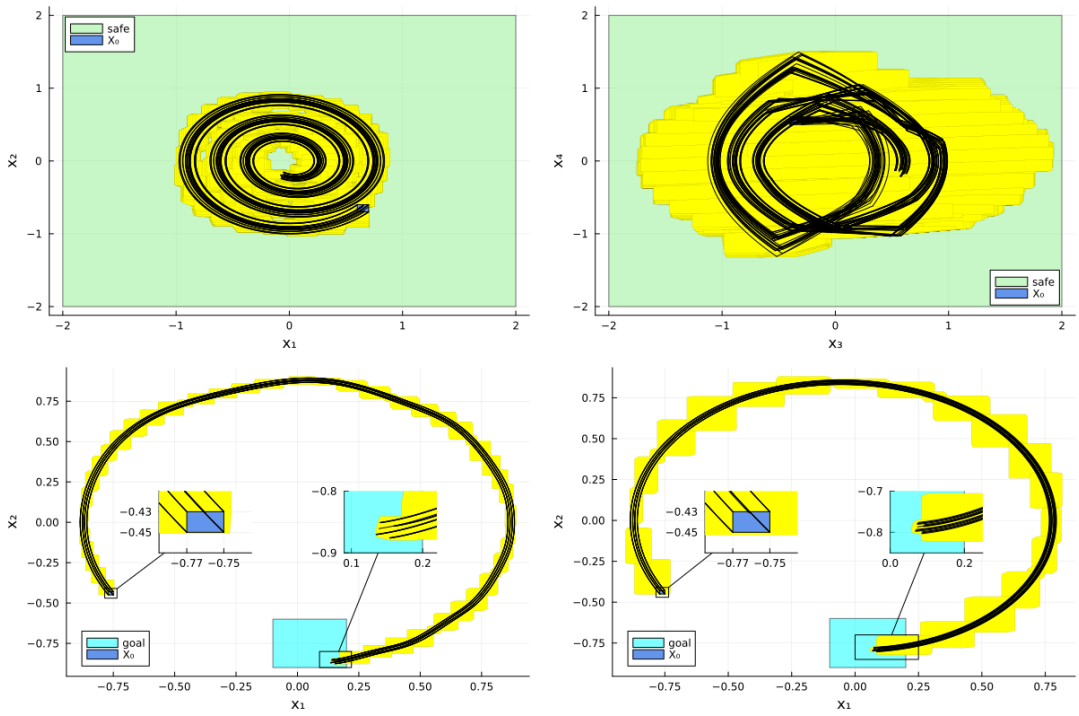


Figure 43: ***JuliaReach.*** Analysis results for the TORA benchmark under the ReLU controller (top), the sigmoid controller (bottom left), and the ReLU/tanh controller (bottom right), respectively. The plots additionally show simulations.

4.4.2 TORA

The TORA benchmark problem has three different controllers. For the ReLU controller, the approximation error is hard to tame for the JuliaReach approach. To maintain enough precision for verification, the initial states are split into $4 \times 4 \times 3 \times 5$ boxes. While each box spawns an independent analysis that could be parallelized, the sequential verification took 184 s. We use the parameters `abstol=3e-2, orderT=3, orderQ=1`. Figure 43 shows the reach sets of all 240 runs, projected to x_1/x_2 and x_3/x_4, respectively.

For the sigmoid and ReLU/tanh controllers, we do not require to split the initial states and use the parameter `abstol=2e-2` instead. The specifications are verified in 1.3 s and 0.2 s, respectively. Figure 43 shows the reach sets, projected to x_1/x_2.

4.4.3 Unicycle

We model the disturbance w as a constant with an uncertain initial value. Simulations show that the target set is reached only at the last moment, so the analysis requires high precision to prove containment of the last reach set. Using the parameters `abstol=1e-1, orderT=3, orderQ=1` and splitting the initial states into $3 \times 1 \times 7 \times 1$ boxes, JuliaReach verifies the specification in 37 s. Figure 44 shows the reach sets of all 21 runs, projected to x_1/x_2 and x_3/x_4, respectively. JuliaReach can evaluate the Taylor polynomial at the time point $t = 10$ (rather than the last time *interval*), which results in a more precise result (as shown in the plots).

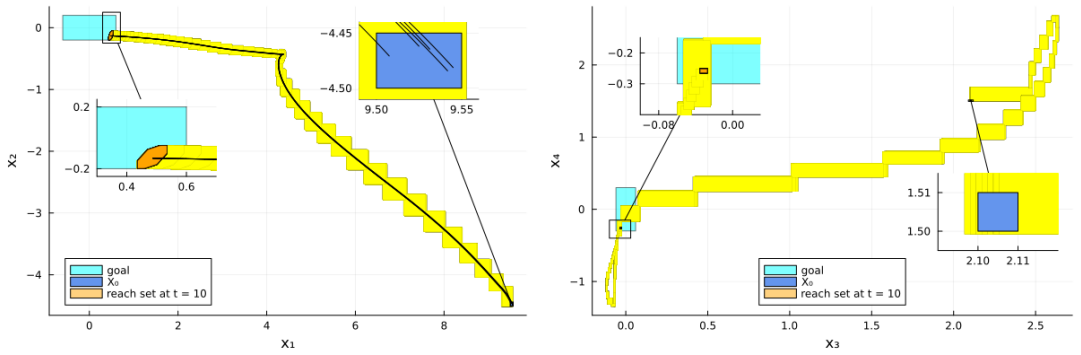


Figure 44: ***JuliaReach.*** Analysis results for the Unicycle benchmark. The orange subset of the last reach set is obtained at time point $t = 10$. The first plot additionally shows simulations.

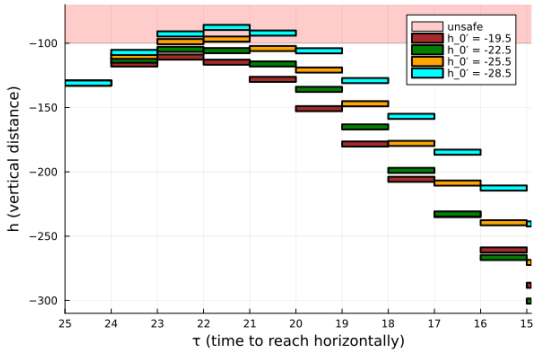


Figure 45: ***JuliaReach.*** Analysis results for the VerticalCAS benchmark.

4.4.4 VerticalCAS

The VerticalCAS benchmark problem differs from the other problems in that it uses multiple controllers and discrete time. There is currently no native support for this setting in JuliaReach; instead, we used a custom algorithm that always chooses the central acceleration. JuliaReach achieves the following results for the different initial values $\dot{h}(0)$. $\dot{h}(0) = -19.5$: verified in 0.01 s; $\dot{h}(0) = -22.5$: verified in 0.001 s; $\dot{h}(0) = -25.5$: falsified in 0.0005 s; $\dot{h}(0) = -28.5$: falsified in 0.0005 s; Figure 45 shows the vertical distances over time.

4.4.5 Single Pendulum

Using the parameters `abstol=1e-9, orderT=5, orderQ=1`, and splitting the initial states into 3×4 non-uniform boxes, JuliaReach verifies the new specification in 14 s. Figure 46 shows the reach sets projected to time and θ.

4.4.6 Double Pendulum

For the less robust controller, using the parameters `abstol=1e-9, orderT=5, orderQ=1` and splitting the initial states into $2 \times 2 \times 3 \times 6$ boxes, JuliaReach verifies the new specification in 25 minutes. Figure 47 shows the reach sets of all 72 runs, projected to θ_1/θ_2 resp. $\dot{\theta}_1/\dot{\theta}_2$.

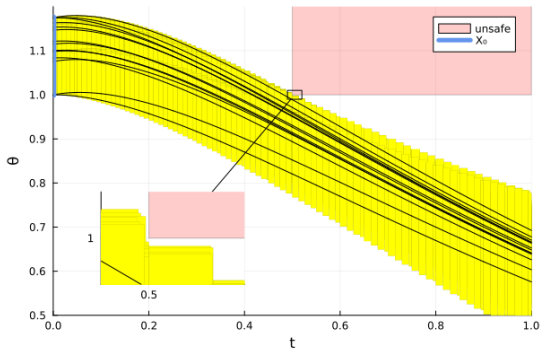


Figure 46: ***JuliaReach.*** Analysis results for the Single Pendulum benchmark. The plot additionally shows simulations.

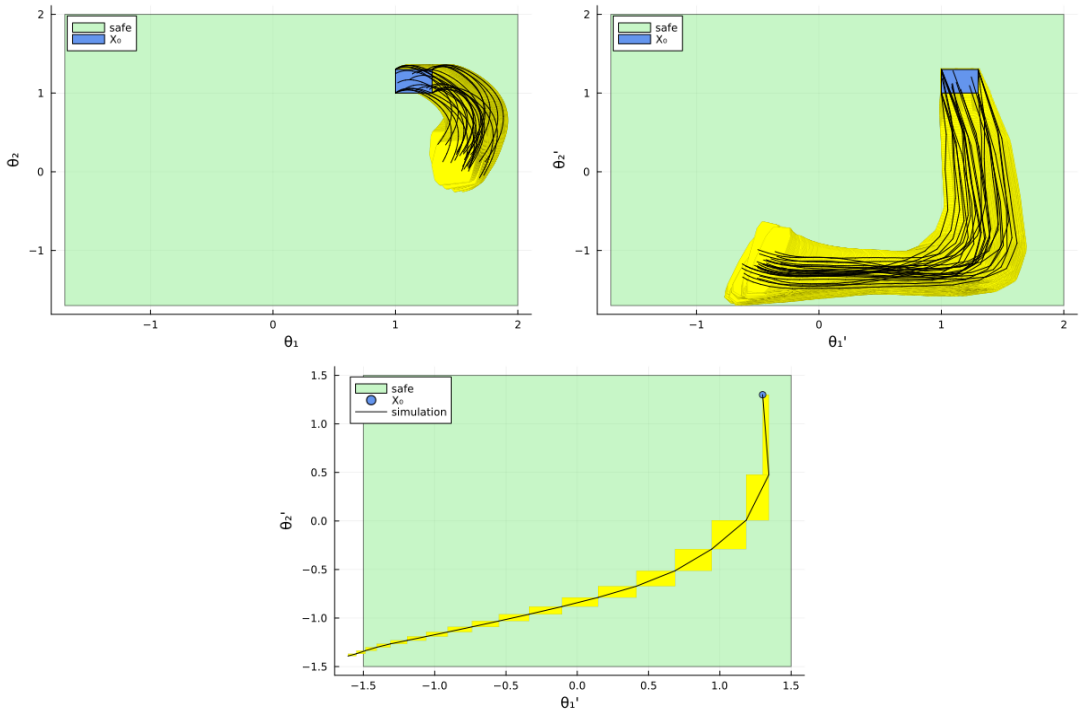


Figure 47: ***JuliaReach.*** Analysis results for the Double Pendulum benchmark under the less robust controller (top) and the more robust controller (bottom). The plots additionally show simulations.

The more robust controller violates the specification; hence it suffices to start the analysis from a subset of the initial states and interrupt when a violation is detected. When starting from the highest value in each dimension, a violation occurs within eighteen control periods. Using the parameters `abstol=1e-2, orderT=3, orderQ=1`, JuliaReach falsifies the specification in 1 s. Figure 47 shows the simulation with a validated reach set around, projected to $\dot{\theta}_1/\dot{\theta}_2$.

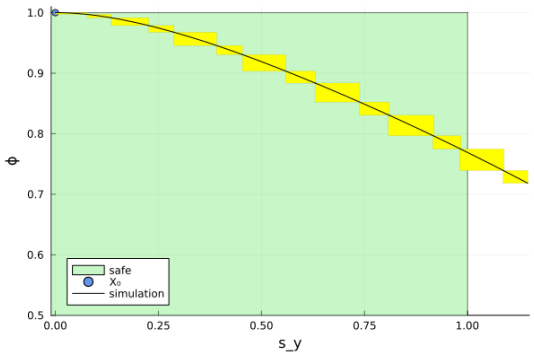


Figure 48: ***JuliaReach.*** Analysis results for the Airplane benchmark until time $t = 0.7$. The plot additionally shows a simulation.

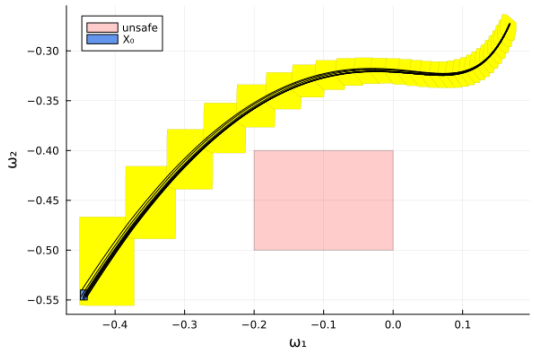


Figure 49: ***JuliaReach.*** Analysis results for the Attitude Control benchmark. The plot additionally shows simulations.

4.4.7 Airplane

This system violates the specification. When starting from the highest coordinate in each dimension, a violation occurs within seven control periods in dimension s_y. Using the parameters `abstol=2e-2, orderT=3, orderQ=1`, JuliaReach falsifies the specification in 2.4 s. Figure 48 shows the simulation with a validated reach set around, projected to y/ϕ.

4.4.8 Attitude Control

Using the parameters `abstol=1e-4, orderT=5, orderQ=1`, JuliaReach verifies the specification in 4.5 s. Figure 49 shows the reach sets projected to ω_1/ω_2.

4.4.9 Quadrotor

Although simulations indicate that the controller is safe, the precision of JuliaReach is not high enough to prove it. The specification can be proven for a smaller initial set $[-0.004, 0.004]^6 \times \{0\}^6$. Using the parameters `abstol=1e-1, orderT=3, orderQ=1`, JuliaReach verifies the specification in 10 s. Figure 50 shows the reach sets, projected to x_3 over time.

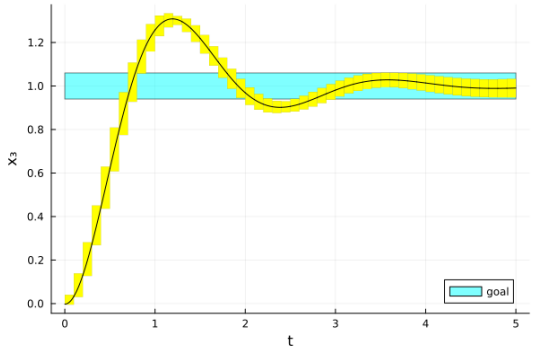


Figure 50: ***JuliaReach.*** Analysis results for the Quadrotor benchmark. The plot additionally shows a simulation.

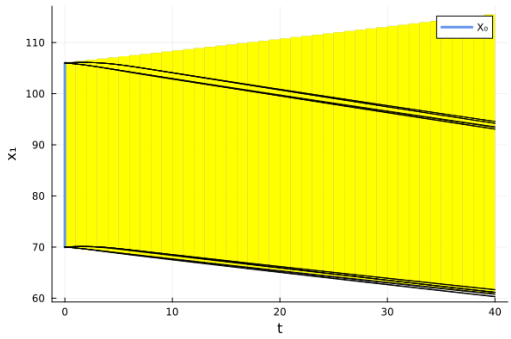


Figure 51: ***JuliaReach.*** Analysis results for the 2D Spacecraft Docking benchmark. The plot additionally shows simulations.

4.4.10 2D Spacecraft Docking

Although simulations indicate that the controller is safe, the precision of JuliaReach is not high enough to prove it. The specification can be proven for a smaller initial set $[70, 106]^2 \times [-0.14, 0.14]^2$. Using the parameters `abstol=5e-1, orderT=3, orderQ=1`, JuliaReach verifies the specification in 4 s. Figure 51 shows the reach sets, projected to x_1 over time. Since the specification is four-dimensional, it cannot be illustrated in the plot.

4.4.11 NAV

JuliaReach cannot verify the standard controller. For the robust controller, using the parameters `abstol=1e-3, orderT=3, orderQ=1`, JuliaReach verifies the specification in 4.5 s. Figure 52 shows the reach sets projected to x_1/x_2.

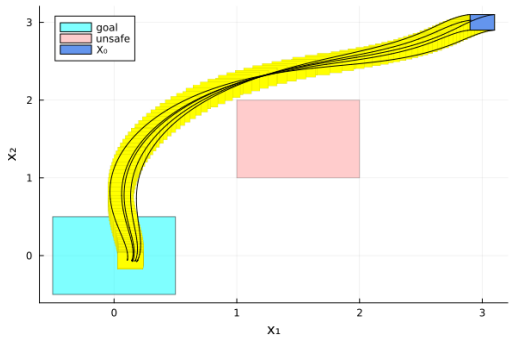


Figure 52: ***JuliaReach.*** Analysis results for the NAV benchmark (robust controller). The plot additionally shows simulations.

4.5 NNV

This subsection presents the results of *JuliaReach*. For each problem, NNV uses slightly different settings, which can be found in the repeatability repository. An overview of the results is given in Tab. 5.

Table 5: ***NNV.*** Overview of results: Verified (✓), falsified (✗), and unknown (**?**).

Benchmark	**Instance**	**Result**	**Time** [s]
ACC	safety	✓	25.000
Airplane	continuous	✗	8.270
AttitudeControl	avoid	?	0.000
Cartpole	reach	?	0.000
Docking	constraint	?	0.000
DoublePendulum	more robust	?	0.000
DoublePendulum	less robust	?	0.000
NAV	standard	?	0.000
NAV	robust	✓	1903.240
QUAD	reach	?	0.000
SinglePendulum	reach	✓	471.310
TORA	remain	✓	19.590
TORA	reach-tanh	✓	282.020
TORA	reach-sigmoid	✓	410.500
Unicycle	reach	?	0.000
VCAS	middle19	✓	3.470
VCAS	middle22	✓	3.420
VCAS	middle25	✗	2.840
VCAS	middle28	✗	3.380
VCAS	worst19	✓	2.720
VCAS	worst22	✗	3.430
VCAS	worst25	✗	3.370
VCAS	worst28	✗	3.220

4.5.1 ACC

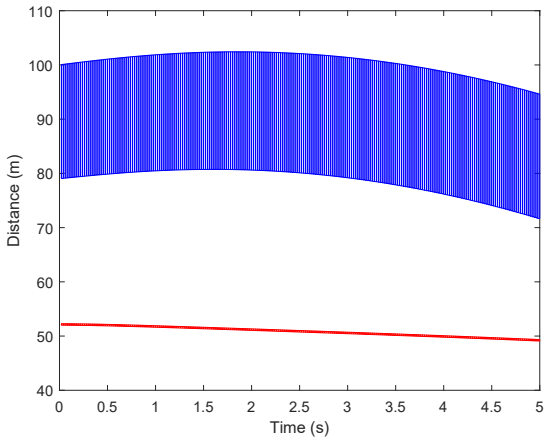


Figure 53: ***NNV.*** Safety analysis results for the ACC benchmark. Distance between cars ($D_{\mathbf{rel}}$) is depicted against the safety distance ($D_{\mathbf{safe}}$).

NNV successfully verifies the safety property $D_{\text{rel}} \geq D_{\text{safe}}$. The results are depicted in Figure 53, which shows the reach sets of D_{rel} and D_{safe}.

4.5.2 TORA

NNV is able to verify all three controllers for the TORA benchmark. NNV partitions the initial sets of the specifications corresponding to the ReLU-tanh and sigmoid controllers, into 4×8×6×4, and 4×4×6×4, respectively. The reach sets are shown in Figure 54.

4.5.3 VerticalCAS

NNV successfully verifies the NMAC safety property for the whole time horizon for each of the cases. There are 5 cases where we prove that the system is unsafe and 3 where the system is safe, which corresponds to [middle, 19.5], [middle, 22.5], and [worst, 19.5]. These results are depicted in Figures 55 and 56.

4.5.4 Single Pendulum

For the single pendulum, NNV successfully verifies the benchmark after partitioning the input set into 35×40 regions. The reach sets are depicted in Figure 57.

4.5.5 Airplane

NNV is able to show that the property is violated by computing the reach sets from a smaller initial region. The results are depicted in Figure 58.

4.5.6 NAV

By partitioning the input set, NNV is able to verify the safety (avoid obstacle) and reach (green goal region) properties of the robust controller, but not of the standard one. The results are depicted in Figure 59.

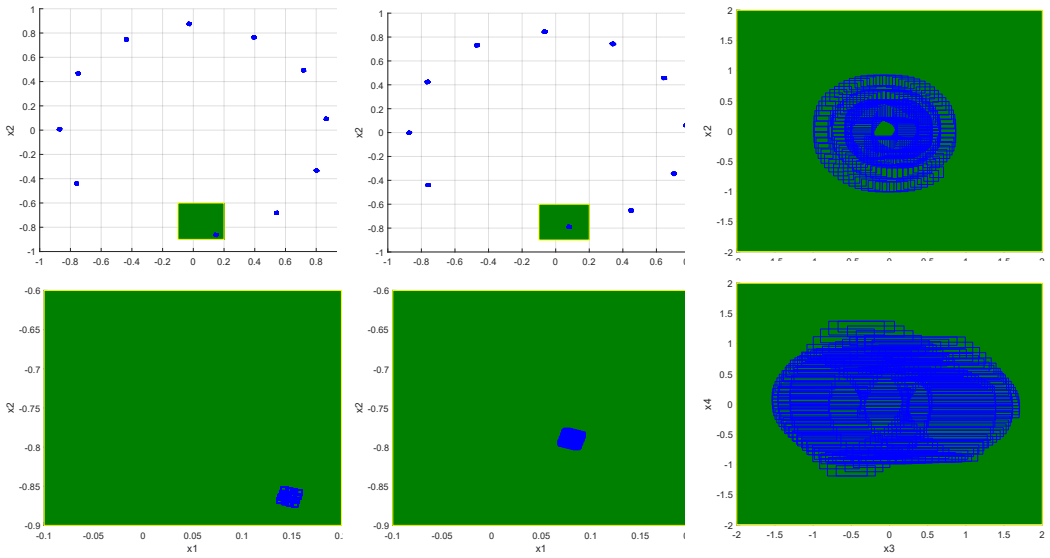


Figure 54: ***NNV.*** Analysis results for the Tora benchmark showing the TORA sets in **blue** and the goal region in **green**. The left figures correspond to the sigmoid controller, the middle two to the ReLU-tanh controller, and the two on the right to the ReLU controller. For the *sigmoid* and *ReLU-tanh* controllers only the reach sets at every control period are shown in the top row. The corresponding zoomed-in pictures of the goal region are depicted in the bottom row.

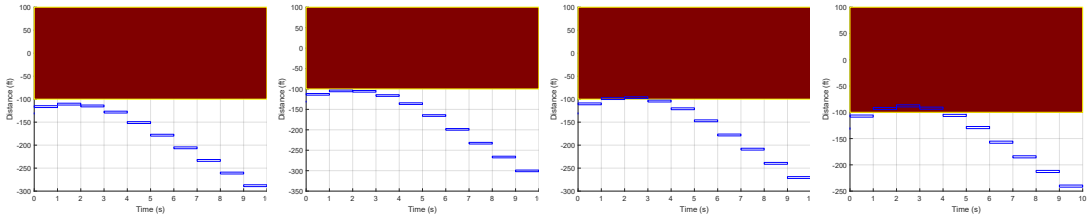


Figure 55: ***NNV.*** Analysis results for the VerticalCAS benchmark, showing the aircraft sets in blue and the unsafe region in red, when selecting the middle acceleration value at each control period

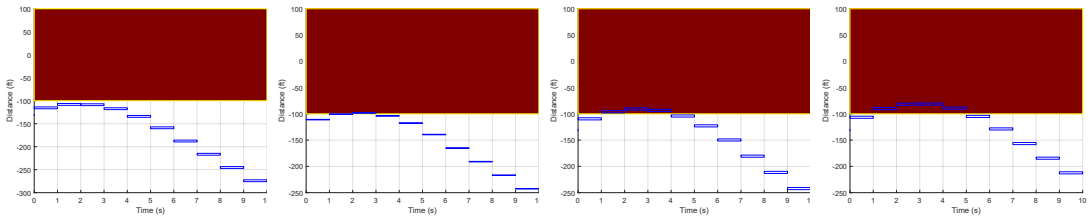


Figure 56: ***NNV.*** Analysis results for the VerticalCAS benchmark, showing the aircraft sets in blue and the unsafe region in red, when selecting the worst possible acceleration value at each control period.

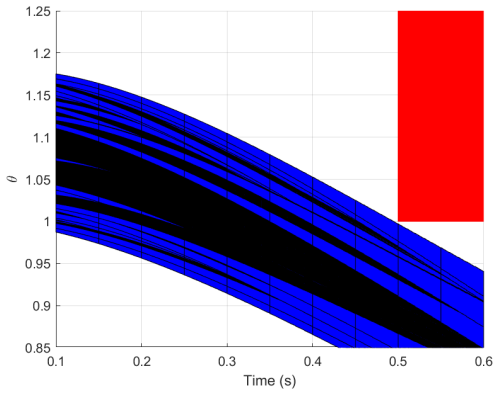


Figure 57: ***NNV.*** Analysis results for the single pendulum benchmark showing the sets in **blue** and the unsafe region in **red**.

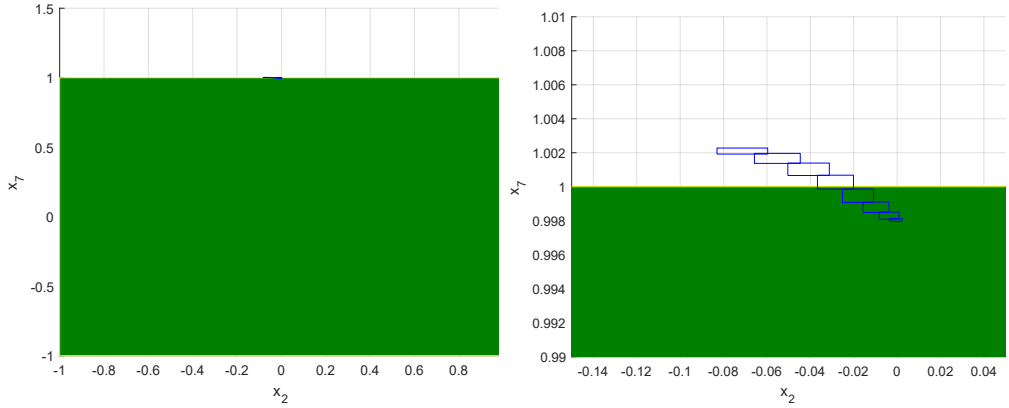
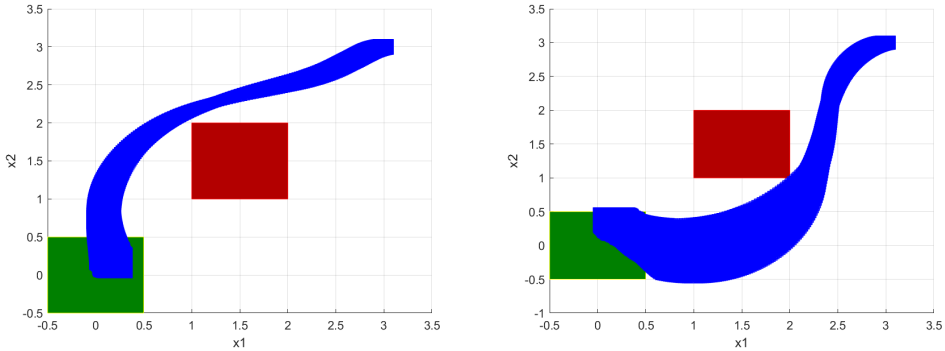


Figure 58: ***NNV.*** Analysis results for the airplane benchmark, showing the reach sets in **blue** and the goal region in **green**.



(a) Robust Controller (V) (b) Standard Controller (?)

Figure 59: ***NNV.*** Analysis results for the NAV benchmark.

5 Category Status and Challenges

Year-over-year comparison. We have observed an immense improvement in the area in the 6 years the competition has been hosted. In 2019, when the competition began, there were only 5 benchmarks, out of which the 3 participants were only able to verify an average of less than 2 benchmarks per tool – including the 2 instances where the controller was modified (ReLU vs tanh) to be supported by one of the tools. Over the next few years (2020-2023), there were a total of 8 different tools that participated in the competition, increasing the complexity of the benchmarks along with the quality improvement of the participating tools. Continuing this trend, this year presented the largest community interest, with 5 tools participating, including 2 newcomers. The results, although not directly comparable to last year's as some existing benchmarks were modified to make them more challenging, demonstrate the lengths the field is growing.

Automatic evaluation. For the second year in a row, we run all tools on the same hardware using docker images for further comparison, using the platform introduced in the 2023 competition. The docker images allow an automatic evaluation of the tools on the submission server, thus, giving researchers immediate feedback on the results of their submission. Running all tools on the same hardware helps to compare the computation time between the tools, however, one has to factor in the efficiency of the programming language of the tools. We can objectively compare computation results and evaluate the improvements made by the tools in this aspect. We observed improvements by up to 1 order of magnitude in computation time in some benchmarks for both JuliaReach and NNV. The submission server specifications are given in Appendix A. In future iterations, we plan to further automate the competition by automatically generating the results section, including tables and figures for all participants.

"Out of the box" comparison. As the community grows and improves, more and more tools are developed. However, as described in this report, many of them require manual tuning of hyperparameters for an optimal computation of the benchmarks. In the future, we would like to add a comparison where participants are not allowed to tune their hyperparameters, provided with unseen benchmarks/specifications, to get a better understanding on the user-friendliness of the tools, as well as on the automated verification capabilities of these.

Statistical approaches. For the first time in the competition, a statistical verification tool has participated: GoTube. The results obtained by this tool are impressive, but the comparison to the sound approaches (all other 4 tools) may not be fair, as GoTube can only ascertain that a benchmark is verified up to a pre-determined percentage (confidence level). In this competition, GoTube uses a confidence level of 90%.

New benchmarks. This year's competition has introduced 2 new benchmarks: NAV and CartPole. The first one demonstrates that that neural networks obtained via adversarial training (*more robust* controller or set-trained instance) help to ease the verification process, in comparison with the *standard* or *less robust* controller, which utilized standard RL training procedures. The latter one demonstrated the challenges that a frequent control step presents, as only GoTube was able to verify it.

Activation Function Types (controllers): For this year's set of benchmarks, all neural network controllers contain one or more of the following activation functions: ReLU, linear, sigmoid, and tanh. The tools have support for the types: linear, piecewise linear (ReLU), and nonlinear activation functions. In the future, we will consider adding other variations of the existing activation functions, such as LeakyReLU or PReLU.

Plant Models: This and last year's competition have considered linear and nonlinear plants, both in discrete and continuous time. In future iterations, we plan to add hybrid automata plants as we look to report a more complete analysis of the participating verification tools. Hybrid automata plants will be especially interesting because combined continuous and discrete dynamics are complex, which is very challenging for current AINNCS verification tools. We will also consider adding neural ODEs as in [53], which will also increase the complexity of the benchmarks and may require a more general encoding of NNCS by most participating tools to verify them.

Neural Network Architectures and Parameterization: The neural network architectures presented in this work are fairly simple. Similar to last year, they have no more than a thousand neurons and no more than 5 hidden layers in their architecture, unlike some of the networks that can be analyzed without the plant. Also, the maximum number of inputs and outputs of the controllers are 12 and 6, respectively, both in the airplane benchmark. Considering the VCAS benchmark, these networks have 9 outputs, although these are translated into a single input to the plant model. However, for some benchmarks, there are still state-space explosion and scalability issues to address in both the neural network controllers and plant analysis. These issues could come from the repeated interaction between the network and the states, as we can observe the CartPole benchmark being challenging for most tools. The high frequency of the control steps may lead to an increased conservativeness of the approaches, preventing most tools from successfully analyzing the benchmark.

Model Formats: Following previous iterations, we have found it more useful and convenient to simply share the plant models in a plain format, such as MATLAB functions, where the participants could easily extract the ODEs. As for the neural network models, we provide them in the ONNX format[5], .mat format[6], and the original format used by the proposer of the benchmark. A few years ago, we began providing the neural network controllers in the ONNX format, as it was very convenient to have a standard exchange format that most of the participating tools supported. However, we have found that there are still discrepancies among the different versions and frameworks these ONNX models were created from (e.g., different input/output transformations are not always supported by every framework as experienced on the *Docking spacecraft* benchmark). Thus, having a standard format easily imported by all participants without local modifications, such as a unified ONNX version, remains a challenge. Initiatives more focused on neural network verification, such as VNN-LIB[7] and VNN-COMP[8], may help toward this goal. In the future, we will consider other options to improve specification

[5]Open Neural Network Exchange: https://github.com/onnx/onnx
[6]Direct input format used by *NNV* without transformation.
[7]http://www.vnnlib.org/
[8]https://github.com/verivital/vnncomp2024

description for benchmarks, as a plain text file is now being used. The creation of a standard format like VNNLIB for AINNCS may help formalize the benchmarks, clearly specifying the input set as well as the AINNCS property (safety, reach, avoid, etc), hopefully lowering the entry barrier for new tools.

6 Conclusion and Outlook

This report presents the results of the sixth ARCH friendly competition for closed-loop systems with neural network controllers. For this edition, five tools have participated and attempted to solve 12 benchmarks: CORA, CROWN-Reach, GoTube, JuliaReach, and NNV. The problems elucidated in this paper are challenging and diverse; the presented results probably provide the most complete assessment of current tools for the safety verification in AINNCS. The report provides a good overview of the intellectual progression of this rapidly growing field, and it is our hope to stimulate the development of efficient and effective methods capable of use in real-world applications. Since its inception, the complexity of the benchmarks has consistently increased along with the capabilities of the participant tools, leading to the most challenging competition and the best verification results thus far, which is a good indicator for this growing and maturing field. This has been achieved thanks to the continuous development and improvements in existing formal verification frameworks, including CORA, JuliaReach, and NNV, and new ones such as CROWN-Reach and GoTube We would also like to encourage other tool developers to consider participating next year, as well as new benchmark proposals are highly welcome. Authors agree that although participation consumes time, we have gained unique insights that have allowed us to improve in each iteration and will allow us to improve in the future, as demonstrated by the yearly improvements by repeating participants. Finally, as we continue to reuse and build upon previous participations, we encourage anyone interested to begin analyzing the benchmarks presented in this iteration, available at: https://github.com/verivital/ARCH-COMP2024. The reports of other categories can be found in the proceedings and on the ARCH website: cps-vo.org/group/ARCH.

7 Acknowledgments

The material presented in this paper is based upon work supported by the National Science Foundation (NSF) through grant numbers 1910017, 2028001, 2220418, 2220426, 2220401, and 2331967 the Defense Advanced Research Projects Agency (DARPA) under contract numbers FA8750-23 C-0518 and FA8750-18-C-0089, and the Air Force Office of Scientific Research (AFOSR) under contract number FA9550-22-1-0019 and FA9550-23-1-0135. Luis Benet acknowledges support from PAPIIT-UNAM project IG-101122. Christian Schilling acknowledges support from the Independent Research Fund Denmark under reference number 10.46540/3120-00041B, DIREC - Digital Research Centre Denmark under reference number 9142-0001B, and the Villum Investigator Grant S4OS under reference number 37819. Tobias Ladner and Matthias Althoff gratefully acknowledge financial support from the project FAI funded by the German Research Foundation (DFG) under project number 286525601. Xiangru Zhong and Huan Zhang acknowledge the support from the AI2050 program at Schmidt Sciences (Grant #G-23-65921).

A Specification of Used Machines

This year, we run all tools on the same hardware using tool-specific docker images. The specifications for the server used for the evaluation are given below. For details on the submission system, we refer to the repeatability report of this year's ARCH competition [44].

- Processor: AMD EPYC 7742 64-Core
- Memory: 995 GB
- OS: Ubuntu 22.04
- Docker: 20.10.21

References

[1] Michael E. Akintunde, Elena Botoeva, Panagiotis Kouvaros, and Alessio Lomuscio. Formal verification of neural agents in non-deterministic environments. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, pages 25–33, 2020.

[2] M. Althoff. An introduction to CORA 2015. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 120–151, 2015.

[3] M. Althoff and D. Grebenyuk. Implementation of interval arithmetic in CORA 2016. In *Proc. of the 3rd International Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 91–105, 2016.

[4] M. Althoff, D. Grebenyuk, and N. Kochdumper. Implementation of Taylor models in CORA 2018. In *Proc. of the 5th International Workshop on Applied Verification for Continuous and Hybrid Systems*, 2018.

[5] M. Althoff, M. Koschi, and S. Manzinger. CommonRoad: Composable benchmarks for motion planning on roads. In *Proc. of the IEEE Intelligent Vehicles Symposium*, pages 719–726, 2017.

[6] S. Bak, S. Bogomolov, T. A. Henzinger, T. T. Johnson, and P. Prakash. Scalable static hybridization methods for analysis of nonlinear systems. In *Proc. of the 19th ACM International Conference on Hybrid Systems: Computation and Control*, pages 155–164, 2016.

[7] Stanley Bak. nnenum: Verification of relu neural networks with optimized abstraction refinement. In *NASA Formal Methods Symposium*, pages 19–36. Springer, 2021.

[8] Stanley Bak, Sergiy Bogomolov, and Taylor T. Johnson. Hyst: A source transformation and translation tool for hybrid automaton models. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, HSCC '15, pages 128–133, New York, NY, USA, 2015. ACM.

[9] Stanley Bak, Changliu Liu, and Taylor Johnson. The second international verification of neural networks competition (vnn-comp 2021): Summary and results. 2021.

[10] Randal W. Beard. Quadrotor dynamics and control. Technical report, 2008.

[11] Luis Benet, Marcelo Forets, David P. Sanders, and Christian Schilling. TaylorModels.jl: Taylor models in Julia and its application to validated solutions of ODEs. In *SWIM*, 2019.

[12] Luis Benet and David P. Sanders. TaylorSeries.jl: Taylor expansions in one and several variables in Julia. *Journal of Open Source Software*, 4(36):1043, 2019.

[13] Luis Benet and David P. Sanders. JuliaDiff/TaylorSeries.jl. https://github.com/JuliaDiff/TaylorSeries.jl, 2021.

[14] Luis Benet and David P. Sanders. JuliaIntervals/TaylorModels.jl. https://github.com/JuliaIntervals/TaylorModels.jl, 2021.

[15] Sergiy Bogomolov, Marcelo Forets, Goran Frehse, Kostiantyn Potomkin, and Christian Schilling. JuliaReach: a toolbox for set-based reachability. In *HSCC*, pages 39–44. ACM, 2019.

[16] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings 25*, pages 258–263. Springer, 2013.

[17] Arthur Clavière, Eric Asselin, Christophe Garion, and Claire Pagetti. Safety verification of neural network controlled systems. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 47–54, 2021.

[18] W. H. CLOHESSY and R. S. WILTSHIRE. Terminal guidance system for satellite rendezvous. *Journal of the Aerospace Sciences*, 27(9):653–658, 1960.

[19] Souradeep Dutta, Xin Chen, and Sriram Sankaranarayanan. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, Montreal, QC, Canada, April 16-18, 2019.*, pages 157–168, 2019.

[20] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Learning and verification of feedback control systems using feedforward neural networks. *IFAC-PapersOnLine*, 51(16):151 – 156, 2018. 6th IFAC Conference on Analysis and Design of Hybrid Systems ADHS 2018.

[21] Michael Everett, Golnaz Habibi, and Jonathan P. How. Efficient reachability analysis of closed-loop systems with neural network controllers. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4384–4390, 2021.

[22] Jiameng Fan, Chao Huang, Wenchao Li, Xin Chen, and Qi Zhu. ReachNN*: A tool for reachability analysis ofneural-network controlled systems. In *to appear on International Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2020.

[23] James Ferlez, Haitham Khedr, and Yasser Shoukry. Fast BATLLNN: fast box analysis of two-level lattice neural networks. In Ezio Bartocci and Sylvie Putot, editors, *HSCC '22: 25th ACM International Conference on Hybrid Systems: Computation and Control, Milan, Italy, May 4 - 6, 2022*, pages 23:1–23:11. ACM, 2022.

[24] Marc Fischer, Christian Sprecher, Dimitar Iliev Dimitrov, Gagandeep Singh, and Martin Vechev. Shared certificates for neural network verification. In Sharon Shoham and Yakir Vizel, editors, *Computer Aided Verification*, pages 127–148, Cham, 2022. Springer International Publishing.

[25] Razvan V Florian. Correct equations for the dynamics of the cart-pole system. *Center for Cognitive and Neural Studies (Coneural), Romania*, page 63, 2007.

[26] Marcelo Forets and Christian Schilling. LazySets.jl: Scalable symbolic-numeric set computations. *Proceedings of the JuliaCon Conferences*, 1(1):11, 2021.

[27] Marcelo Forets and Christian Schilling. The inverse problem for neural networks. In *AISoLA*, volume 14380 of *LNCS*, pages 241–255. Springer, 2023.

[28] Eric Goubault and Sylvie Putot. Rino: Robust inner and outer approximated reachability of neural networks controlled systems. In Sharon Shoham and Yakir Vizel, editors, *Computer Aided Verification*, pages 511–523, Cham, 2022. Springer International Publishing.

[29] Sophie Gruenbacher, Ramin Hasani, Mathias Lechner, Jacek Cyranka, Scott A. Smolka, and Radu Grosu. On the verification of neural odes with stochastic guarantees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11525–11535, 2021.

[30] Sophie A. Gruenbacher, Mathias Lechner, Ramin Hasani, Daniela Rus, Thomas A. Henzinger, Scott A. Smolka, and Radu Grosu. Gotube: Scalable statistical verification of continuous-depth models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 6755–6764, 2022.

[31] G. W. Hill. Researches in the lunar theory. *American Journal of Mathematics*, 1(1):5–26, 1878.

[32] Chao Huang, Jiameng Fan, Xin Chen, Wenchao Li, and Qi Zhu. POLAR: A polynomial arithmetic framework for verifying neural-network controlled systems. In *To appear on International Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2022.

[33] Radoslav Ivanov, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. *CoRR*, abs/1811.01828, 2018.

[34] M. Jankovic, D. Fontaine, and P. V. Kokotovic. Tora example: cascade- and passivity-based control designs. *IEEE Transactions on Control Systems Technology*, 4(3):292–297, May 1996.

[35] Taylor T. Johnson, Diego Manzanas Lopez, Luis Benet, Marcelo Forets, Sebasti\'an Guadalupe, Christian Schilling, Radoslav Ivanov, Taylor J. Carpenter, James Weimer, and Insup Lee. ARCH-COMP21 category report: Artificial intelligence and neural network control systems (AINNCS) for continuous and hybrid systems plants. In Goran Frehse and Matthias Althoff, editors, *8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21)*, volume 80 of *EPiC Series in Computing*, pages 90–119. EasyChair, 2021.

[36] Taylor T Johnson, Diego Manzanas Lopez, Patrick Musau, Hoang-Dung Tran, Elena Botoeva, Francesco Leofante, Amir Maleki, Chelsea Sidrane, Jiameng Fan, and Chao Huang. Arch-comp20 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants. In Goran Frehse and Matthias Althoff, editors, *ARCH20. 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20)*, volume 74 of *EPiC Series in Computing*, pages 107–139. EasyChair, 2020.

[37] K. D. Julian and M. J. Kochenderfer. A reachability method for verifying dynamical systems with deep neural network controllers. *CoRR*, abs/1903.00520, 2019.

[38] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In Rupak Majumdar and Viktor Kunčak, editors, *Computer Aided Verification*, pages 97–117, Cham, 2017. Springer International Publishing.

[39] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, David L. Dill, Mykel J. Kochenderfer, and Clark Barrett. The marabou framework for verification and analysis of deep neural networks. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, pages 443–452, Cham, 2019. Springer International Publishing.

[40] Niklas Kochdumper and Matthias Althoff. Sparse polynomial zonotopes: A novel set representation for reachability analysis. *IEEE Transactions on Automatic Control*, 66(9):4043–4058, 2020.

[41] Niklas Kochdumper, Christian Schilling, Matthias Althoff, and Stanley Bak. Open-and closed-loop neural network verification using polynomial zonotopes. In *NASA Formal Methods Symposium*, pages 16–36. Springer, 2023.

[42] Lukas Koller, Tobias Ladner, and Matthias Althoff. End-to-end set-based training for neural network verification. *arXiv preprint arXiv:2401.14961*, 2024.

[43] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *CoRR*, abs/1607.02533, 2016.

[44] Tobias Ladner. ARCH-COMP24 repeatability evaluation report. 2024.

[45] Tobias Ladner and Matthias Althoff. Automatic abstraction refinement in neural network verification using sensitivity analysis. *HSCC'23: Proceedings of the 26th International Conference on Hybrid Systems: Computation and Control*, 2023.

[46] Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, and Mykel J. Kochenderfer. Algorithms for verifying deep neural networks. *Foundations and Trends in Optimization*, 4(3-4):244–404, 2021.

[47] Diego Manzanas Lopez, Matthias Althoff, Luis Benet, Xin Chen, Jiameng Fan, Marcelo Forets, Chao Huang, Taylor T Johnson, Tobias Ladner, Wenchao Li, Christian Schilling, and Qi Zhu. ARCH-COMP22 category report: Artificial intelligence and neural network control systems (AINNCS) for continuous and hybrid systems plants. In Goran Frehse, Matthias Althoff, Erwin Schoitsch, and Jeremie Guiochet, editors, *Proceedings of 9th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22)*, volume 90 of *EPiC Series in Computing*, pages 142–184. EasyChair, 2022.

[48] Diego Manzanas Lopez, Matthias Althoff, Marcelo Forets, Taylor T. Johnson, Tobias Ladner, and Christian Schilling. ARCH-COMP23 category report: Artificial intelligence and neural network control systems (AINNCS) for continuous and hybrid systems plants. In Goran Frehse and Matthias Althoff, editors, *10th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH23)*, EPiC Series in Computing, pages 89–125. EasyChair, 2023.

[49] Diego Manzanas Lopez, Sung Woo Choi, Hoang-Dung Tran, and Taylor T. Johnson. NNV 2.0: The neural network verification tool. In *35th International Conference on Computer-Aided Verification (CAV)*, July 2023.

[50] Diego Manzanas Lopez, Patrick Musau, Hoang-Dung Tran, Souradeep Dutta, Taylor J. Carpenter, Radoslav Ivanov, and Taylor T. Johnson. Arch-comp19 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants. In Goran Frehse and Matthias Althoff, editors, *ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems*, volume 61 of *EPiC Series in Computing*, pages 103–119. EasyChair, 2019.

[51] Amir Maleki and Chelsea Sindrane. Benchmark examples for ainncs-2020, 2020.

[52] Diego Manzanas Lopez, Taylor T. Johnson, Stanley Bak, Hoang-Dung Tran, and Kerianne L. Hobbs. Evaluation of neural network verification methods for air-to-air collision avoidance. *Journal of Air Transportation*, 31(1):1–17, 2023.

[53] Diego Manzanas Lopez, Patrick Musau, Nathaniel Hamilton, and Taylor Johnson. Reachability analysis of a general class of neural ordinary differential equation. In *Proceedings of the 20th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2022), Co-Located with CONCUR, FMICS, and QEST as part of CONFEST 2022.*, Warsaw, Poland, September 2022.

[54] MathWorks. *Adaptive Cruise Control System* block. https://www.mathworks.com/help/mpc/ref/adaptivecruisecontrolsystem.html, 2018.

[55] Mark Niklas Müller, Christopher Brix, Stanley Bak, Changliu Liu, and Taylor T. Johnson. The third international verification of neural networks competition (vnn-comp 2022): Summary and results, 2022.

[56] K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27, March 1990.

[57] Jorge A. Pérez-Hernández and Luis Benet. PerezHz/TaylorIntegration.jl. https://github.com/PerezHz/TaylorIntegration.jl, 2021.

[58] S. Prajna, P.A. Parrilo, and A. Rantzer. Nonlinear control synthesis by convex optimization. volume 49, pages 310–314, 2004.

[59] S. Joe Qin and Thomas A. Badgwell. An overview of nonlinear model predictive control applications. In Frank Allgöwer and Alex Zheng, editors, *Nonlinear Model Predictive Control*, pages 369–392, Basel, 2000. Birkhäuser Basel.

[60] Umberto J. Ravaioli, James Cunningham, John McCarroll, Vardaan Gangal, Kyle Dunlap, and Kerianne L. Hobbs. Safe reinforcement learning benchmark environments for aerospace control systems. In *2022 IEEE Aerospace Conference (AERO)*, pages 1–20, 2022.

[61] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.

[62] Stuart Russell, Daniel Dewey, and Max Tegmark. Research Priorities for Robust and Beneficial Artificial Intelligence. *AI Magazine*, 36(4):105, 2015.

[63] Christian Schilling, Marcelo Forets, and Sebastián Guadalupe. Verification of neural-network control systems by integrating Taylor models and zonotopes. In *AAAI*, pages 8169–8177. AAAI Press, 2022.

[64] Zhouxing Shi, Qirui Jin, J Zico Kolter, Suman Jana, Cho-Jui Hsieh, and Huan Zhang. Formal verification for neural networks with general nonlinearities via branch-and-bound. *2nd Workshop on Formal Verification and Machine Learning*, 2023.

[65] Malcolm D. Shuster. Survey of attitude representations. *Journal of the Astronautical Sciences*, 41(4):439–517, October 1993.

[66] Chelsea Sidrane and Mykel J. Kochenderfer. OVERT: Verification of nonlinear dynamical systems with neural network controllers via overapproximation. *Safe Machine Learning workshop at ICLR*, 2019.

[67] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T. Vechev. Fast and effective robustness certification. In *NeurIPS*, pages 10825–10836, 2018.

[68] Peter Stone, Rodney Brooks, Erik Brynjolfsson, Ryan Calo, Oren Etzioni, Greg Hager, Julia Hirschberg, Shivaram Kalyanakrishnan, Ece Kamar, Kevin Leyton-Brown, David C. Parkes, William Press, AnnaLee Saxenian, Julie Shah, Milind Tambe, and Astro Teller. "artificial intelligence and life in 2030." one hundred year study on artificial intelligence: Report of the 2015-2016 study panel, 2016.

[69] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.

[70] Hoang-Dung Tran, Stanley Bak, Weiming Xiang, and Taylor T. Johnson. Verification of deep convolutional neural networks using imagestars. In *32nd International Conference on Computer-Aided Verification (CAV)*. Springer, July 2020.

[71] Hoang-Dung Tran, Feiyang Cai, Manzanas Lopez Diego, Patrick Musau, Taylor T. Johnson, and Xenofon Koutsoukos. Safety verification of cyber-physical systems with reinforcement learning control. *ACM Trans. Embed. Comput. Syst.*, 18(5s), October 2019.

[72] Hoang Dung Tran, SungWoo Choi, Tomoya Yamaguchi, Bardh Hoxha, and Danil Prokhorov. Verification of recurrent neural networks using star reachability. In *The 26th ACM International Conference on Hybrid Systems: Computation and Control (HSCC)*, May 2023.

[73] Hoang-Dung Tran, Diago Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T. Johnson. Star-based reachability analysis of deep neural networks. In Maurice H. ter Beek, Annabelle McIver, and José N. Oliveira, editors, *Formal Methods – The Next 30 Years*, pages 670–686. Springer International Publishing, 2019.

[74] Hoang-Dung Tran, Neelanjana Pal, Patrick Musau, Xiaodong Yang, Nathaniel P. Hamilton, Diego Manzanas Lopez, Stanley Bak, and Taylor T. Johnson. Robustness verification of semantic segmentation neural networks using relaxed reachability. In *33rd International Conference on Computer-Aided Verification (CAV)*. Springer, July 2021.

[75] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *32nd International Conference on Computer-Aided Verification (CAV)*, July 2020.

[76] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. *Advances in Neural Information Processing Systems*, 2021.

[77] Manuel Wendl, Lukas Koller, Tobias Ladner, and Matthias Althoff. Training verifiably robust agents using set-based reinforcement learning. *arXiv preprint*, 2024.

[78] Weiming Xiang, Patrick Musau, Ayana A. Wild, Diego Manzanas Lopez, Nathaniel Hamilton, Xiaodong Yang, Joel A. Rosenfeld, and Taylor T. Johnson. Verification for machine learning, autonomy, and neural networks survey. *CoRR*, abs/1810.01989, 2018.

[79] Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems*, 33:1129–1141, 2020.

[80] Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In *International Conference on Learning Representations*, 2020.

[81] Huan Zhang, Shiqi Wang, Kaidi Xu, Linyi Li, Bo Li, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. General cutting planes for bound-propagation-based neural network verification. *Advances in Neural Information Processing Systems*, 2022.

[82] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. *Advances in neural information processing systems*, 31, 2018.