EPiC Series in Computing

Volume 74, 2020, Pages 107–139

ARCH20. 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20)

ARCH-COMP20 Category Report: Artificial Intelligence and Neural Network Control Systems (AINNCS) for Continuous and Hybrid Systems Plants

Taylor T. Johnson[1], Diego Manzanas Lopez[1], Patrick Musau[1], Hoang-Dung Tran[1], Elena Botoeva[2], Francesco Leofante[2], Amir Maleki[3], Chelsea Sidrane[3], Jiameng Fan[4], Chao Huang[5]

[1] Vanderbilt University
Nashville, TN
{diego.manzanas.lopez, patrick.musau, dung.h.tran, taylor.johnson}@vanderbilt.edu
[2] Imperial College London
London, UK
{f.leofante, e.botoeva}@imperial.ac.uk
[3] Stanford University
Stanford, CA
{amir.maleki, csidrane}@stanford.edu
[4] Boston University
Boston, MA
jmfan@bu.edu
[5] Northwestern University
Evanston, IL
chao.huang@northwestern.edu

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 more 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 2020. In the second edition of this AINNCS category at ARCH-COMP, four tools have been applied to solve seven different benchmark problems, (in alphabetical order): NNV, OVERT, ReachNN*, and VenMAS. This report is a snapshot of the current landscape of tools and the types of benchmarks for which these tools are suited. Due to the diversity of problems, lack of a shared hardware platform, and the early stage of the competition, we are not ranking tools in terms of performance, yet the presented results probably provide the most complete assessment of current tools for safety verification of NNCS.

1 Introduction

Neural Networks (NNs) have demonstrated an impressive ability for solving complex problems in numerous application domains [32]. In fact, the success of these models in contexts such as adaptive control, non-linear system identification [22], image and pattern recognition, function approximation, and machine translation, has stimulated the creation of technologies that are directly impacting our everyday lives [26], and has led researchers to believe that these models possess the power to revolutionize a diverse set of arenas [24].

Despite these achievements, there have been reservations in utilizing them within high-assurance systems for a variety of reasons, such as their susceptibility to unexpected and errant behavior caused by slight perturbations in their inputs [20]. In a study by Szegedy et al. [27], the authors demonstrated that by carefully applying a hardly perceptible modification to an input image, one 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 [32]. As a result, there has been a large research effort directed towards obtaining an explicit understanding of neural network behavior.

Neural networks are often viewed as "black boxes," whose underlying operation is often incomprehensible, but the last several years have witnessed a large number of promising white-box verification methods proposed towards reasoning about the correctness of their behavior. However, it has been demonstrated that neural network verification is an NP-complete problem [19], and while current state-of-the-art verification methods have been able to deal with small networks, they are incapable of dealing with the complexity and scale of networks used in practice ([21, 12, 6]). Additionally, while in recent years there has been a large amount of work focused on verifying pre-/post-conditions for neural networks in isolation, reasoning about the behavior of their usage 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 [10]. Furthermore, this report aims to provide readers with a perspective of the intellectual progression of this rapidly growing field and stimulate the development of efficient and effective methods capable of use in real-life applications.

Disclaimer The presented report of the ARCH 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*. We would like to stress that each tool has unique strengths—not all of the specificities can be highlighted within a single report. To reach a consensus in 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.

Specifically, this report summarizes results obtained in the 2020 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. Participating tools are summarized in Sec. 2. Please, see [32] for further details on these and additional tools. The results of our selected benchmark problems are shown in Sec. 3 and are obtained on the tool developers' own machines. Thus, one has to factor in the computational power of the processors used, summarized in Appendix A, as well as the efficiency of the programming language of the tools. The architecture of the closed-loop systems we will evaluate is depicted in Figure 1, where the input to the NN controller is additionally sampled.

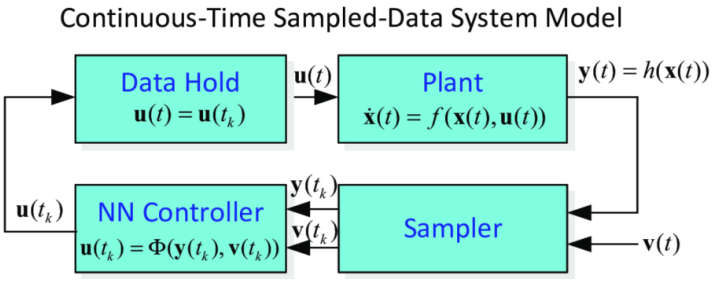

Figure 1: Closed-loop architecture of the benchmarks to be verified.

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 not possible 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.

2 Participating Tools

We present a brief overview of all the participating tools in this friendly competition. The tools are NNV, OVERT, ReachNN*, and VenMAS. The tools participating in this AINNCS category *Artificial Intelligence / Neural Network Control Systems in Continuous and Hybrid Systems Plants* are introduced subsequently in alphabetical order.

NNV NNV (Neural Network Verification Tool) [30, 28, 29, 35, 37, 34, 33, 31, 36, 1] is a Matlab toolbox that implements reachability analysis methods for neural network verification, with a particular focus on applications of closed-loop neural network control systems in autonomous cyber-physical systems. 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

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

feed-forward and convolutional neural networks. The star-set based algorithm is naturally parallelizeable, which allowed NNV to be designed to perform efficiently on multi-core platforms. Additionally, in the event that a particular safety property is violated, NNV can be used to construct and visualize the complete set of counterexample inputs for a neural network. Using NNV in combination with HySST [8, 7] and CORA[3, 4, 5] allows for the verification of closed-loop neural network control systems with nonlinear plant dynamics. The tool along with all of the relevant experiments and publications can be found at https://github.com/verivital/nnv.

OVERT OVERT [25] is a tool for verifying discrete-time, closed-loop systems with neural network controllers. Specifically, OVERT is designed for use with ReLU neural network controllers and discrete-time dynamical systems of any kind. The key technical insight of OVERT is to overapproximate smooth non-linearities in the dynamics with piecewise linear functions, and to employ a relational abstraction to compose the overapproximations. This allows OVERT to be sound, modulo finite precision arithmetic. The resulting problem may then be reasoned about using any automated reasoning tool or constraint solving tool that is capable of efficiently handling piecewiselinear nonlinearities. The results presented here use an MIP encoding with the Gurobi solver [15]. OVERT may be used to produce concrete reach sets or to directly prove properties without explicitly producing the reachable set.

ReachNN* ReachNN*, [16, 13, 14] is a tool that verifies the reach-avoid specification for neural-network controlled systems. The theoretical foundation of ReachNN* is the use of Bernstein polynomials to approximate any Lipschitz-continuous neural-network controller, with provable approximation error bounds [16]. Then the resulting polynomial systems can be verified by current tools, e.g. Flow* [9]. Benefiting from that Bernstein polynomials are universal approximators, ReachNN* has the capability to handle the networks with different types of activation functions. The error bound analysis relies on the partition of the state space and is accelerated by GPU-based parallel computing [14]. Additionally, when the given neural-network controller is hard to verify, ReachNN* also features optional controller re-synthesis via a technique called *verification-aware knowledge distillation* (KD) to obtain a verification-friendly neural-network controller by reducing the Lipschitz constant of the original controller [13]. ReachNN* is available at https://github.com/JmfanBU/ReachNNStar.

VenMAS VenMAS [2] is a tool for verification of closed-loop systems with neural network components. VenMAS supports closed-loop system consisting of an environment and a number of agents, who interact with each other and with the environment and thus update their state. It is assumed that the agents' logic is partially or fully implemented using ReLU-based neural networks. The environment's transition function is assumed to be a piecewise linear function, and can be in particular implemented via a ReLU-based neural network.

VenMAS takes as input a specification of the agent, of the environment and property against which the closed-loop system is to be verified. The property should be a temporal formula expressible in bounded Computational Tree Logic (CTL). Venus reduces verification to a Mixed-Integer Linear Programming (MILP) feasibility problem and relies on Gurobi to solve the latter. The tool is available for download at https://vas.doc.ic.ac.uk/software/neural/.

3 Benchmarks

For the competition, we have selected seven benchmarks. A few of them, such as the TORA benchmark, are presented with several different controllers that can be analyzed. Additionally, some benchmarks from last year's edition of the competition were kept. These benchmarks are the Adaptive Cruise Controller (ACC), and Sherlock's benchmarks 9 (TORA) and 10. We have also added Vertical Collision Avoidance System (VCAS), Single Pendulum and Double Pendulum, and Airplane benchmarks. We now describe these benchmarks in no particular order and we have made them readily available online.[2]

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. The neural network computes optimal control actions while satisfying safe distance, velocity, and acceleration constraints using model predictive control (MPC) [23]. 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 as follows:

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

where x_i is the position, v_i is the velocity, γ_i is the acceleration of the car, a_i is the acceleration control input applied to the car, and $u = 0.0001$ is the friction control where $i \in \{\text{ego, lead}\}$. For this benchmark we have developed four neural network controllers with 3, 5, 7, and 10 hidden layers of 20 neurons each, although we only evaluate the one with 5 layers. All of them have the same number of inputs (v_{set}, T_{gap}, v_{ego}, D_{rel}, v_{rel}), and one output (a_{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_{lead} = -2. We want to check whether there is a collision in the following 5 seconds. Formally, this safety specification of the system can be expressed as $D_{\text{rel}} = x_{\text{lead}}$ - $x_{\text{ego}} \geq D_{\text{safe}}$, where $D_{\text{safe}} = D_{\text{default}} + T_{\text{gap}} \times v_{\text{ego}}$, and $T_{\text{gap}} = 1.4$ seconds and $D_{\text{default}} = 10$. The initial conditions are: $x_{\text{lead}}(0) \in$ [90,110], $v_{\text{lead}}(0) \in$ [32,32.2], $\gamma_{\text{lead}}(0) = \gamma_{\text{ego}}(0) = 0$, $v_{\text{ego}}(0) \in$ [30, 30.2], $x_{\text{ego}} \in$ [10,11]. A control period of 0.1 seconds is used.

3.2 Sherlock-Benchmark-9 (TORA)

This benchmark is that of a TORA (translational oscillations by a rotational actuator) [10, 17]. The model is that of a cart attached to a wall with a spring, and is free to move on friction-less surface. The cart itself has a weight attached to an arm inside it, which is free to rotate about an axis. This serves as the control input, in order to stabilize the cart at $\mathbf{x} = \mathbf{0}$. The model is 4 dimensional system, given by the following equations :

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

[2] https://github.com/verivital/ARCH-COMP2020

A neural network controller was trained for this system, using data-driven model predictive controller proposed in [11]. The trained network had 3 hidden layers, with 100 neurons in each layer making a total of 300 neurons. Note that the output of the neural network $f(\mathbf{x})$ needs to be normalized in order to obtain u, namely $u = f(\mathbf{x}) - 10$. The sampling time for this controller was $1s$.

Specification The verification problem here is that of safety. 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 stay within the box $\mathbf{x} \in [-2, 2]^4$, for a time window of $20s$.

3.3 Sherlock-Benchmark-10 (Unicycle Car Model)

This benchmark is that of a unicycle model of a car [10]. It models the dynamics of a car involving 4 variables, specifically the x and y coordinates on a 2 dimensional plane, as well as velocity magnitude (speed) and steering angle.

$$\dot{x_1} = x_4 \cos(x_3), \dot{x_2} = x_4 \sin(x_3), \dot{x_3} = u_2, \dot{x_4} = u_1 + w, \tag{3}$$

where w is a bounded error in the range $[-1e-4, 1e-4]$. A neural network controller was trained for this system, using a model predictive controller as a "demonstrator" or "teacher". The trained network has 1 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 was $0.2s$.

Specification The verification problem here is that of reachability. 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 $x_4 \in [1.5, 1.51]$, it is required to prove that the system reaches 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 $10s$.

3.4 VCAS Benchmark

This benchmark is a closed-loop variant of *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 [18]. Once every second, VerticalCAS issues vertical climb rate advisories to the ownship pilot to avoid a near mid-air collision (NMAC). Near mid-air collisions are regions in which 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 climbrate, and
- τ, the seconds 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.

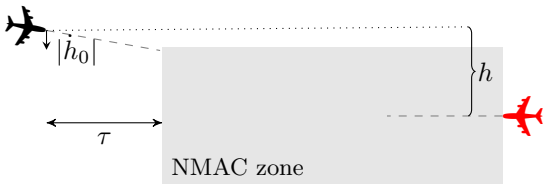

Figure 2: VerticalCAS encounter geometry

There are 9 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 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 (1), 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: (2) DNC: $\{-\frac{g}{3}, -\frac{7g}{24}, -\frac{g}{4}\}$; (3) DND: $\{\frac{g}{4}, \frac{7g}{24}, \frac{g}{3}\}$; (4) DES1500: $\{-\frac{g}{3}, -\frac{7g}{24}, -\frac{g}{4}\}$; (5) CL1500: $\{\frac{g}{4}, \frac{7g}{24}, \frac{g}{3}\}$; (6) SDES1500: $\{-\frac{g}{3}\}$; (7) SCL1500: $\{\frac{g}{3}\}$; (8) SDES2500: $\{-\frac{g}{3}\}$; (9) 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 account for all possible choices of acceleration efficiently. 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 that will take the ownship closer to or less far apart from 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 $(h(t+1), \dot{h}_0(t+1), \tau(t+1), \text{adv}(t+1))$ can be computed as follows:

$$\begin{aligned} h(t+1) &= h - \dot{h}_0 \Delta\tau - 0.5\ddot{h}_0 \Delta\tau^2 \\ \dot{h}_0(t+1) &= \dot{h}_0 + \ddot{h}_0 \Delta\tau \\ \tau(t+1) &= \tau - \Delta\tau \\ \text{adv}(t+1) &= \text{adv}' \end{aligned}$$

where $\Delta\tau = 1$.

Specification For this benchmark the aim is to verify that the ownship is 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 as follows: $h(0) \in [-133, -129]$, $\dot{h}_0(0) \in \{-19.5, -22.5, -25.5, -28.5\}$, $\tau(0) = 25$ and adv(0) = COC.

3.5 Single Pendulum Benchmark

This is the classical inverted pendulum environment. 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 exists with a coefficient of c. The governing equation of motion can be obtained as:

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

where θ is the angle that link makes with the upward vertical axis, and $\dot{\theta}$ is the angular velocity. The state vector is:

$$[\theta, \dot{\theta}] \tag{5}$$

Controllers are trained using *behavior cloning*, a supervised learning approach for training controllers. Here, a neural network is trained to replicate expert demonstrations. We initially generate a set of *expert* control inputs for trajectories originating from different initial states of the system. *Expert* control inputs are defined as those that lead the system to reach to its goal state in finite time. The expert control inputs are generated using optimal control techniques. Specifically, we have used an implementation of LQR (Linear Quadratic Regulator) and iLQR(iterative LQR) control. The code for these implementations, as well training procedures are provided.

The continuous-time equations of motion may be written as a series of first order ODEs where $x_1 = \theta$ and $x_2 = \dot{\theta}$:

$$\dot{x}_1 = x_2 \tag{6a}$$

$$\dot{x}_2 = \frac{g}{L} \sin x_1 + \frac{1}{mL^2} \left(T - c\, x_2 \right) \tag{6b}$$

The difference equations for the discrete time version of the system are obtained by using forward Euler integration:

$$x_{1_{t+1}} = x_{1_t} + \dot{x}_{1_t} \Delta t \tag{7a}$$

$$x_{2_{t+1}} = x_{2_t} + \dot{x}_{2_t} \Delta t \tag{7b}$$

The model involves several parameters, as follows.

$$m = 0.5, L = 0.5, c = 0., g = 1.0$$

The controller timestep (and dynamics timestep for discrete time) for `controller single pendulum` is $\Delta t = 0.05$. The initial set is

$$[\theta, \dot{\theta}] = [1.0, 1.2] \times [0.0, 0.2].$$

Specification The discrete-time safety specification is: $\forall n_t : 10 \leq n_t \leq 20, \theta \in [0.0, 1.0]$. The continuous-time safety specification is $10 \leq t \leq 20$, $\theta \in [0, 1]$.

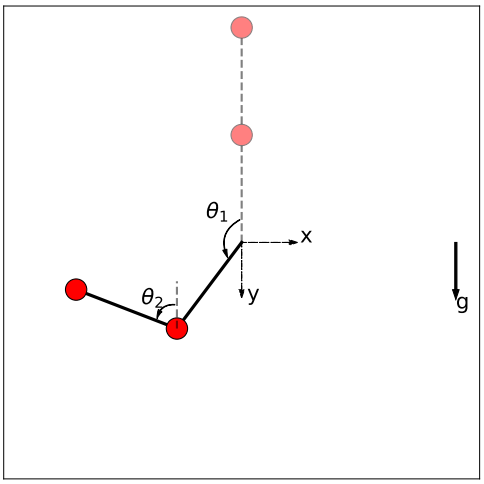

Figure 3: Inverted double pendulum. The goal is keep the pendulum upright (dashed schematics)

3.6 Double Pendulum Benchmark

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:

$$2\ddot{\theta}_1 + \ddot{\theta}_2 \cos(\theta_2 - \theta_1) - \dot{\theta}_2^2 \sin(\theta_2 - \theta_1) - 2\frac{g}{L}\sin\theta_1 + \frac{c}{mL^2}\dot{\theta}_1 = \frac{1}{mL^2}T_1 \tag{8a}$$

$$\ddot{\theta}_1 \cos(\theta_2 - \theta_1) + \ddot{\theta}_2 + \dot{\theta}_1^2 \sin(\theta_2 - \theta_1) - \frac{g}{L}\sin\theta_2 + \frac{c}{mL^2}\dot{\theta}_2 = \frac{1}{mL^2}T_2 \tag{8b}$$

where θ_1 and θ_2 are the angles that links make with the upward vertical axis (seeFigure 3). The state is:

$$[\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2] \tag{9}$$

The angular velocity and acceleration of links are denoted with $\dot{\theta}_1, \dot{\theta}_2, \ddot{\theta}_1$ and $\ddot{\theta}_2$ and g is the gravitational acceleration.

The controllers for the double pendulum benchmark are obtained using the same methods as the controllers for the single pendulum benchmark: behavior cloning from LQR and iLQR trajectories. The continuous-time equations of motion may be written as a series of first order ODEs where $x_1 = \theta_1$, $x_2 = \theta_2$, $x_3 = \dot{\theta}_1$ and $x_4 = \dot{\theta}_2$. See eq 11. The difference equations for the discrete time version of the system are obtained by using forward Euler integration:

$$x_{i_{t+1}} = x_{i_t} + \dot{x}_{i_t}\Delta t \text{ for } i \in [1, 2, 3, 4]. \tag{10}$$

The model involves several parameters:

$$m = 0.5, L = 0.5, c = 0., g = 1.0.$$

Specification This benchmark has two controllers, each with slightly different specifications. Use `controller double pendulum less robust` with $\Delta t = 0.05$. The initial set is

$$[\theta_1, \theta_2, \dot{\theta}_1\dot{\theta}_2] = [1.0, 1.3]^4.$$

The safety specification is

$$\forall n_t \leq 20, [\theta_1, \theta_2, \dot{\theta}_1 \dot{\theta}_2] \in [-1.0, 1.7]^4.$$

The continuous time version is the same as the discrete-time version, with $n_t = 20$ corresponding to $t_{max} = 0.05 * 20 = 1sec$.

Use `controller double pendulum more robust` with $\Delta t = 0.02$. The initial set is

$$[\theta_1, \theta_2, \dot{\theta}_1 \dot{\theta}_2] = [1.0, 1.3]^4.$$

The safety specification is

$$\forall n_t \leq 20, [\theta_1, \theta_2, \dot{\theta}_1 \dot{\theta}_2] \in [-0.5, 1.5]^4.$$

The continuous time version is the same as the discrete-time version, with $n_t = 20$ corresponding to $t_{max} = 0.02 * 20 = 0.4sec$.

These equations are generated with Matlab:

$$\dot{x}_1 = x_3 \tag{11a}$$

$$\dot{x}_2 = x_4 \tag{11b}$$

$$\dot{x}_3 = \frac{\bigstar \left({x_3}^2 \sin(x_1 - x_2) - \bigstar \left(\frac{g \sin(x_1)}{L} - \frac{{x_4}^2 \sin(x_1 - x_2)}{2} + \frac{T_1 - c\, x_3}{2\, L^2\, m} \right) + \frac{g \sin(x_2)}{L} + \frac{T_2 - c\, x_4}{L^2\, m} \right)}{2 \left(\frac{\cos^2(x_1 - x_2)}{2} - 1 \right)} \tag{11c}$$

$$- \frac{{x_4}^2 \sin(x_1 - x_2)}{2} + \frac{g \sin(x_1)}{L} + \frac{T_1 - c\, x_3}{2\, L^2\, m} \tag{11d}$$

$$\dot{x}_4 = - \frac{{x_3}^2 \sin(x_1 - x_2) - \bigstar \left(\frac{g \sin(x_1)}{L} - \frac{{x_4}^2 \sin(x_1 - x_2)}{2} + \frac{T_1 - c\, x_3}{2\, L^2\, m} \right) + \frac{g \sin(x_2)}{L} + \frac{T_2 - c\, x_4}{L^2\, m}}{\frac{\cos^2(x_1 - x_2)}{2} - 1}, \tag{11e}$$

where $\bigstar = \cos(x_1 - x_2)$.

3.7 Airplane Benchmark

The airplane example consists of a dynamical system that is a simple model of a flying airplane. It can be visualized in Figure 4. The state is:

$$[x, y, z, u, v, w, \phi, \theta, \psi, r, p, q] \tag{12}$$

where (x, y, z) is the position of the C.G., (u, v, w) 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:

$$\dot{u} = - g \sin\theta + \frac{F_x}{m} - qw + rv \tag{13a}$$

$$\dot{v} = g \cos\theta \sin\phi + \frac{F_y}{m} - ru + pw \tag{13b}$$

$$\dot{w} = g \cos\theta \cos\phi + \frac{F_z}{m} - pv + qu \tag{13c}$$

$$I_x \dot{p} + I_{xz} \dot{r} = M_x - (I_z - I_y) qr - I_{xz} pq \tag{13d}$$

$$I_y \dot{q} = M_y - I_{xz} \left(r^2 - p^2 \right) - (I_x - I_z) pr \tag{13e}$$

$$I_{xz} \dot{p} + I_z \dot{r} = M_z - (I_y - I_x) qp - I_{xz} rq. \tag{13f}$$

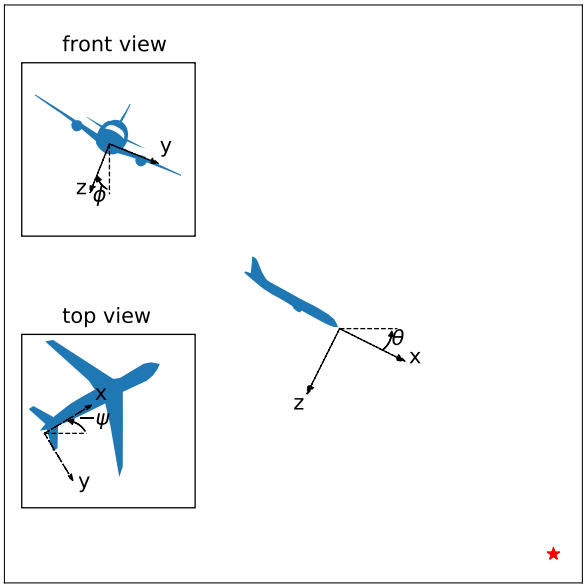

Figure 4: The airplane example.

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 controls parameters include three force components F_x, F_y and F_z and three moment components M_x, M_y, M_z. Notice that for simplicity we have assumed the aerodynamic forces are absorbed in the F's. In addition to these six equations, we have six additional kinematic equations:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix} = \begin{bmatrix}\cos\psi & -\sin\psi & 0\\ \sin\psi & \cos\psi & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos\theta & 0 & \sin\theta\\ 0 & 1 & 0\\ -\sin\theta & 0 & \cos\theta\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\phi & -\sin\phi\\ 0 & \sin\phi & \cos\phi\end{bmatrix}\begin{bmatrix}u\\ v\\ w\end{bmatrix} \tag{14}$$

and

$$\begin{bmatrix}\phi\\ \theta\\ \psi\end{bmatrix} = \begin{bmatrix}1 & \tan\theta\sin\phi & \tan\theta\cos\phi\\ 0 & \cos\phi & -\sin\phi\\ 0 & \sec\theta\sin\phi & \sec\theta\cos\phi\end{bmatrix}\begin{bmatrix}p\\ q\\ r\end{bmatrix} \tag{15}$$

As in the pendulum benchmarks, controllers are trained for the airplane problem using behavior cloning from LQR and iLQR trajectories.

The state is defined to be $[x, y, z, u, v, w, \phi, \theta, \psi, r, p, q]$ and the derivatives that form the system of continuous time equations are specified in eqs 13,14,15. The difference equations for the discrete time version of the system are obtained by using the following mapping: $[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}] = [x, y, z, u, v, w, \phi, \theta, \psi, r, p, q]$ and forward Euler integration:

$$x_{i_{t+1}} = x_{i_t} + \dot{x}_{i_t}\Delta t \text{ for } i \in [1, 2, 3, 4]. \tag{16}$$

The system involves the following model parameters:

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

Use the `controller airplane` with $\Delta t = 0.1$. The initial set is

$$x = y = z = r = p = q = 0, [u, v, w, \phi, \theta, \psi] = [0.0, 1.0]^6.$$

Specification The safety specification is

$$\forall n_t : n_t \leq 20, y \in [-0.5, 0.5], [\phi, \theta, \psi] = [-1.0, 1.0]^3.$$

The continuous time version is the same as the discrete-time version, with $n_t = 20$ corresponding to $t_{max} = .1 * 20 = 2sec$.

4 Verification Results

For each of the participating tools, we obtained verification results for each of the proposed benchmarks. Reachable sets are shown for those methods that are able to construct them.

4.1 NNV

We present the results utilizing *NNV* on each of the benchmarks. The experiments were performed on a machine with the following specifications: Intel Core i7-8750H CPU@2.20GHz, 12 core processor, 32 GB memory, and 64-bit Windows 10.

4.1.1 ACC

For the ACC benchmark, we present results using a neural network controller with 101 neurons (5-by-20) with ReLU activation functions, and use a time horizon of 5 seconds, as shown in Figure 5. We observe that the safety of the car is not guaranteed, as the intersection of the actual (relative) and safe distance is not empty throughout the 5-second time horizon.

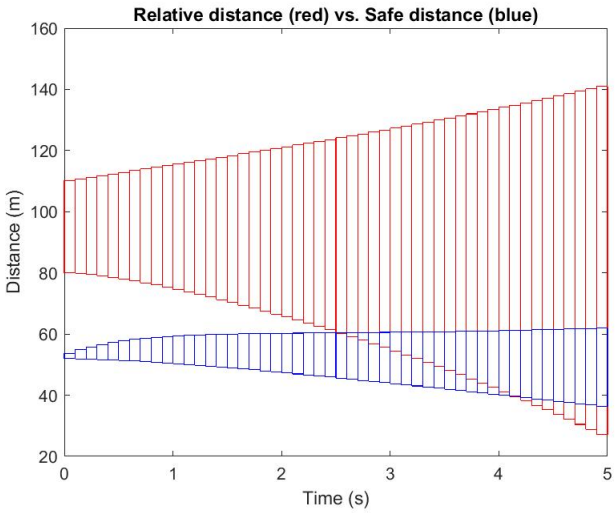

Figure 5: ***NNV.*** Reachability analysis results of the ACC benchmark using a controller with 5 hidden layers (ReLU) of 20 neurons each.

4.1.2 Sherlock-Benchmark-9

The results displayed in Figure 6 were obtained after 229.89 seconds of computation. The result is unknown due to the use of over-approximation analysis. Although dimensions the first three dimensions satisfy the safety property, the fourth does not.

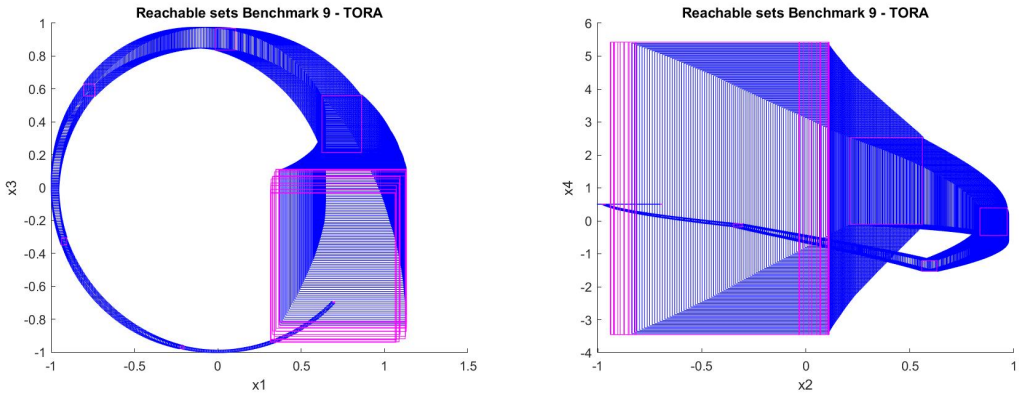

Figure 6: ***NNV.*** Reachability analysis results of the TORA Sherlock benchmark 9. All dimensions are shown: Dimensions 1 and 2 are plotted in the left figure, dimensions 3 and 4 in the right.

4.1.3 Sherlock-Benchmark-10

On this benchmark, NNV runs out of memory due to the number of computation steps and splits needed in the computation of the plant's reachable sets. We have plotted a graph with the first two dimensions in Figure 7. One can see how the size of the reachable sets increases as time progresses. This growth leads to NNV running out of memory.

4.1.4 TORA Heterogeneous

The experimental results for both of the controllers considered in our analysis are similar. Both result in a verification result of unknown due to the over-approximate scheme utilized for nonlinear functions. In both cases, we initialize the analysis with a smaller initial set than initially proposed in order to promote faster computation. The experiments demonstrate that due to the over-approximation scheme, the system may reach the proposed area, but it is not guaranteed. The results for the ReluTanh controller are shown in Figure 8. These sets were obtained after 221 seconds.

The results for the sigmoid controller are shown in Figure 9 and the computation time was 31.50 seconds.

4.1.5 VCAS

For the VCAS Benchmark, there are two scenarios that we investigate. The first one is the worst-case scenario, where we show that after 3 steps, the system is unsafe under all possible initial velocities. The results are shown in Figure 10.

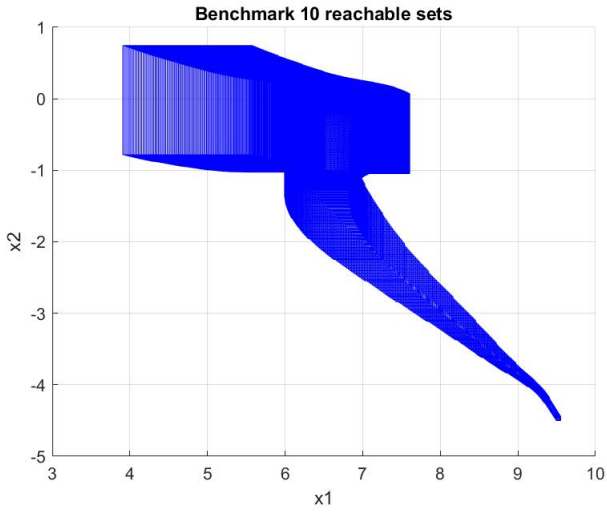

Figure 7: ***NNV.*** Reachability Analysis results for benchmark 10 (Unicycle).

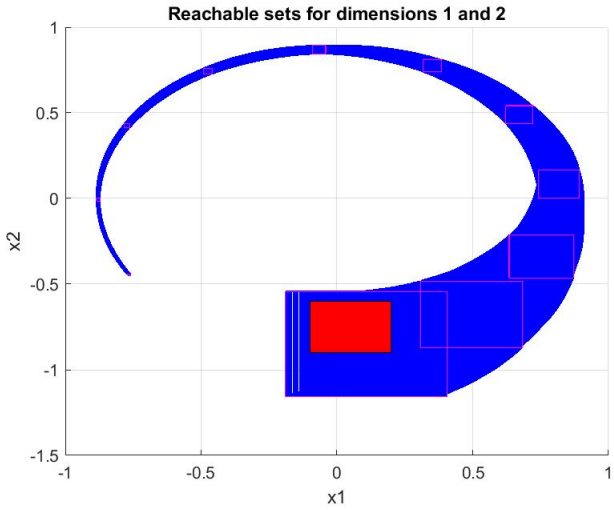

Figure 8: ***NNV.*** Reachability analysis results of TORA with ReLU-Tanh controller.

For the second case, choosing the middle acceleration when possible, we also show that they system is unsafe after a few steps for all possible initial velocities. Results are shown in Figure 11.. The computation time for each case was about 1-2 seconds.

4.1.6 Single Pendulum

The verification result for the single pendulum system is unsafe, as shown in Figure 12. The angle θ is not within the bounds [0.0, 1.0] after 10 time steps. The total computation time is 65.5 seconds.

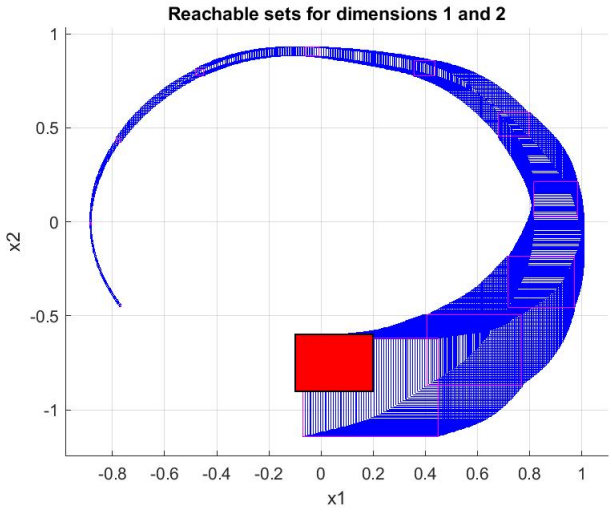

Figure 9: ***NNV.*** Reachability analysis results of TORA with sigmoid controller.

4.1.7 Double Pendulum

The results for the system with the less robust controller are shown in Figure 13. One can see that dimensions 1, 2 and 4 still satisfy the safety property, but dimension 3 does not due to the over-approximate analysis. The total computation time is 27.94 seconds.

The results for the system with the more robust controller are shown in Figure 14. This is a similar case to the previous controller as dimension 3 does not satisfy the safety property while the other still do. The total computation time is 13.64 seconds.

4.1.8 Airplane

The airplane benchmark is a high-dimensional nonlinear benchmark, which makes the analysis very computationally expensive. The analysis utilized a total of 501.26 seconds for 13 control steps. After these steps, each dimension satisfies the safety property except for dimension 2, which due to the over-approximation in the reachable sets, reaches the -1 and 1 safety boundaries. The resulting reachable sets are shown in Figure 15.

4.2 ReachNN*

We present the results of all the benchmarks proposed using *ReachNN**. All experiments are performed on a desktop with 12-core 3.60 GHz Intel Core i7 and NVIDIA GeForce RTX 2060.

4.2.1 ACC

In the ACC example, we verified safety for the initial condition $x_{lead}(0) \in [90, 110], x_{ego}(0) \in [10, 11]$, $v_{\text{lead}}(0) \in [32, 32.2]$, $\gamma_{\text{lead}}(0) = \gamma_{\text{ego}}(0) = 0$, $v_{\text{ego}}(0) \in [30,\ 30.2]$. With the original controller, ReachNN cannot directly verify the safety property up to 10 steps. The reachable set becomes extremely large. Thus, we invoke the knowledge distillation component in ReachNN* and distilled a new network with a much smaller Lipschitz constant 1.2. Then, we verify the

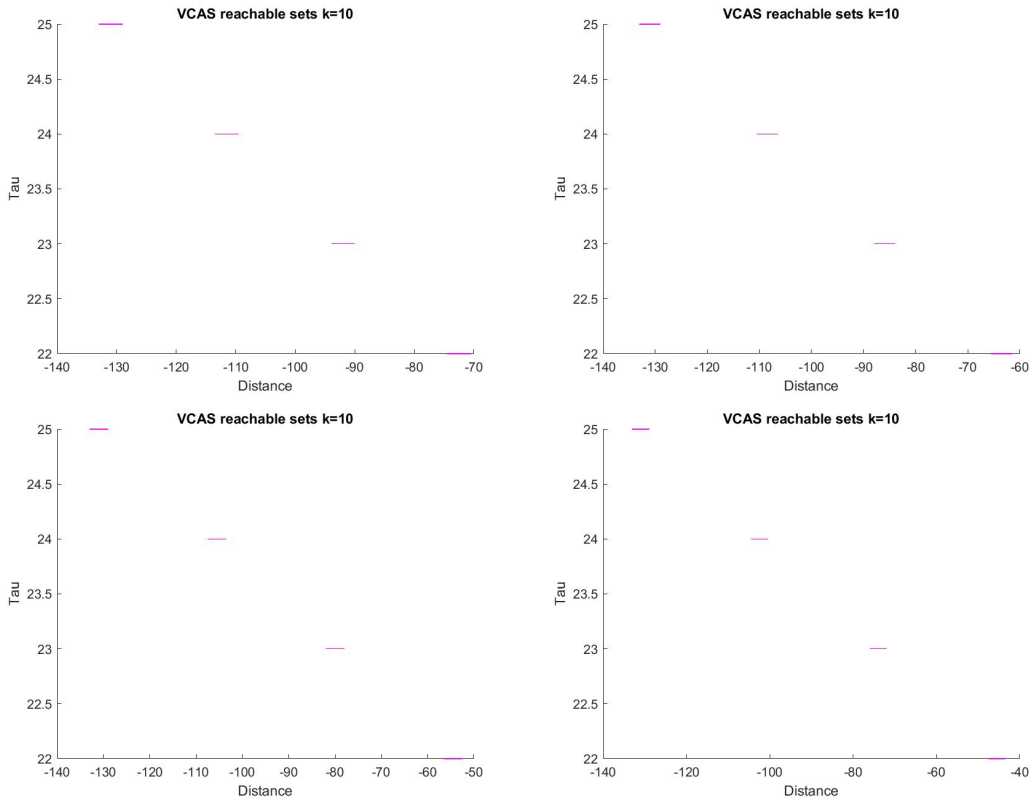

Figure 10: ***NNV.*** VCAS results when we choose the middle acceleration. The simulations were stopped before 10 steps since the safety property was violated.

NNCS with this newly distilled network. The reachable sets for the positions of the two cars are shown in Figure 16. Notice that the safety property can be verified with the newly distilled network while the original NNCS cannot be verified directly. The reachable set computation takes 529 seconds.

4.2.2 Airplane

There is no result for this benchmark due to the high input dimension. ReachNN* can theoretically handle this benchmark. However, the 12-dimensional input introduces large polynomials, which causes out-of-memory error when doing reachability analysis by Flow* [9].

4.2.3 Sherlock-Benchmark-10-Unicycle

There is no result for this benchmark since this benchmark requires multiple control inputs. Current version of ReachNN* only supports the neural-network controller with a single output. Future version of ReachNN* will include this feature.

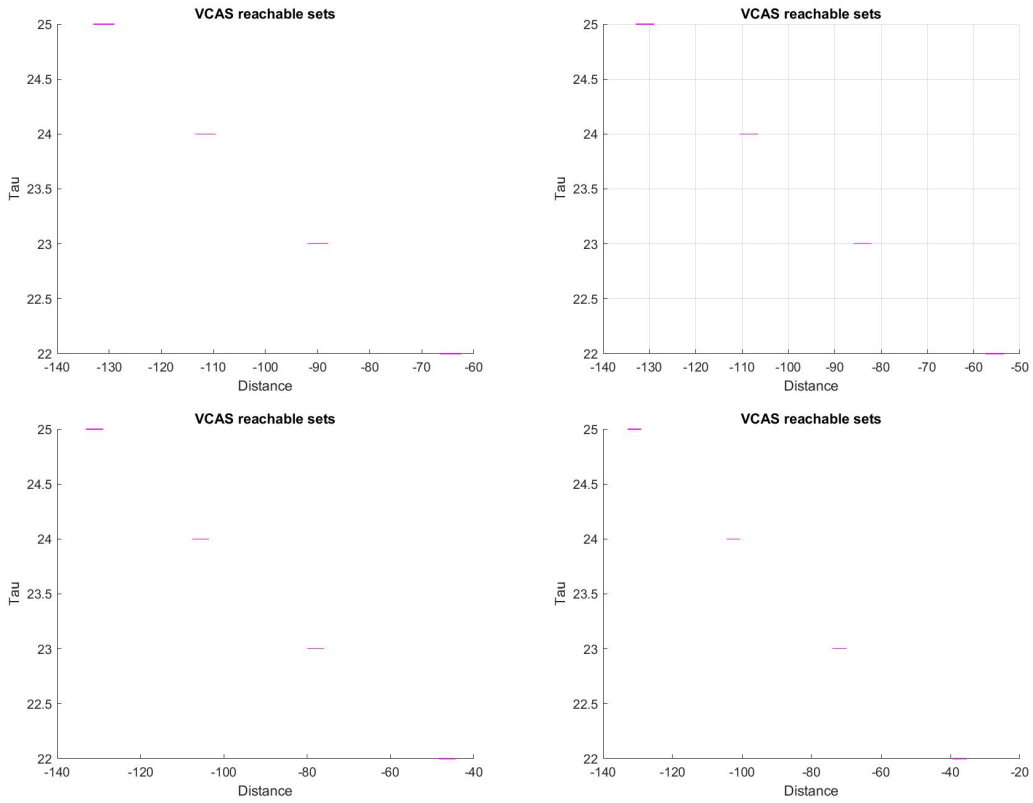

Figure 11: ***NNV.*** VCAS results when we choose the worst possible acceleration. The simulations were stopped before 10 steps since the safety property was violated.

4.2.4 Sherlock-Benchmark-9

In the Sherlock-Benchmark-9 example, ReachNN* is not able to verify the safety specification for the original controller due to the large estimated approximation error. Thus, we leverage the knowledge distillation module to obtain the a new network controller with a smaller Lipschitz constant. The system with the new controller can be successfully verified by ReachNN* in 1226 seconds. The reachable sets can be found in Figure 17. We can find that knowledge distillation component in ReachNN* can effectively obtain a more verification-friendly neural-network controller when the original design is difficult to verify.

4.2.5 Double-Pendulum

There is no result for this benchmark.

4.2.6 Single-Pendulum

There is no result for this benchmark.

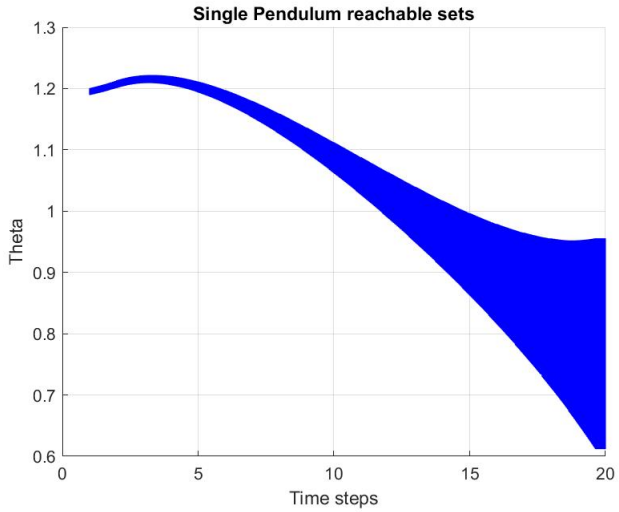

Figure 12: ***NNV.*** Single pendulum reachable sets.

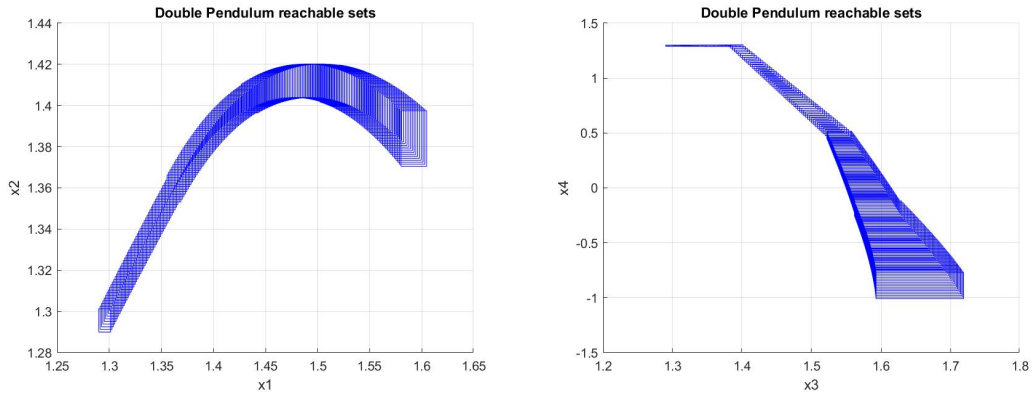

Figure 13: ***NNV.*** Double pendulum with less robust controller.

4.2.7 Tora-Heterogeneous

The main difference between the Tora-Heterogeneous example and the Sherlock-Benchmark-9 example is that, the neural-network controller contains ReLU and tanh activations simultaneously in the Tora-Heterogeneous example. Benefiting from the universal approximation of Bernstein polynomials, ReachNN* can well handle heterogeneous neural networks and successfully verifies the Tora-Heterogeneous example. The reachable sets computed by ReachNN* are shown in Figure 18. ReachNN* finishes the computation in 1583 seconds.

4.2.8 VCAS

There is no result for this benchmark due to the discrete dynamics. The current version of ReachNN* only supports continuous systems.

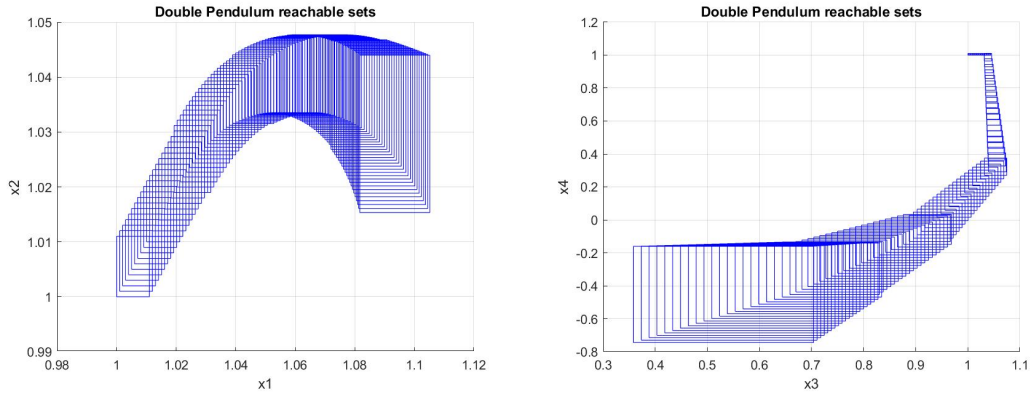

Figure 14: ***NNV.*** Double pendulum with more robust controller.

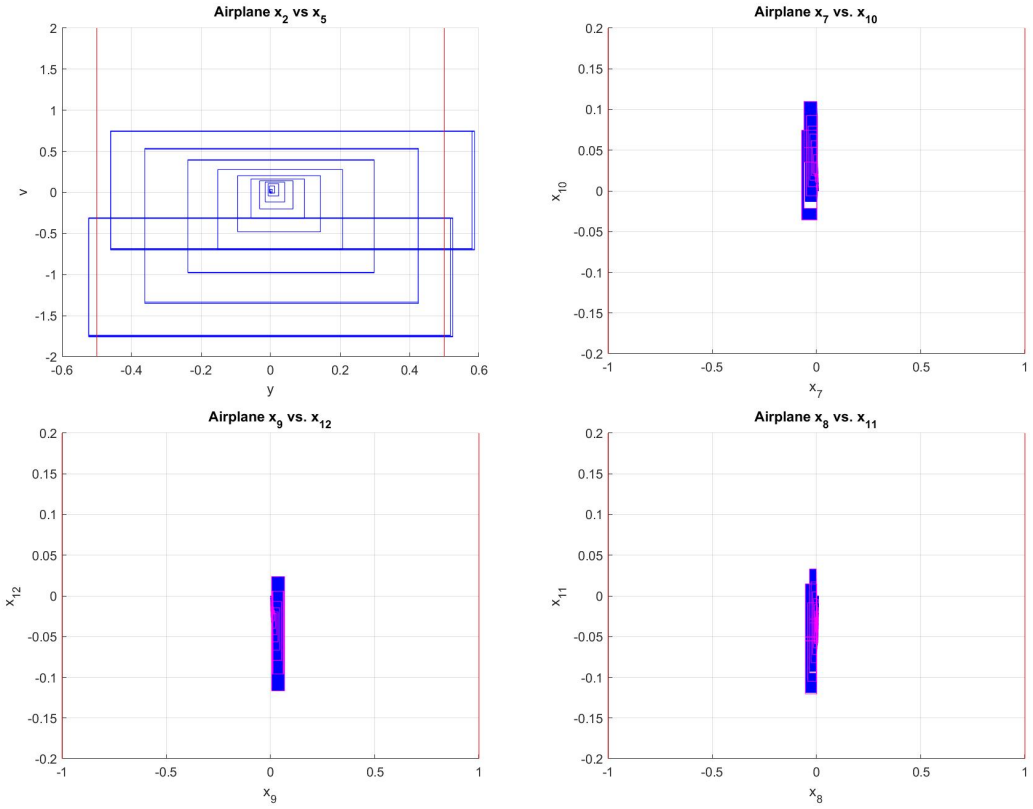

Figure 15: ***NNV.*** Airplane reachable sets. In red, the safety boundaries. In blue, the system's reachable sets.

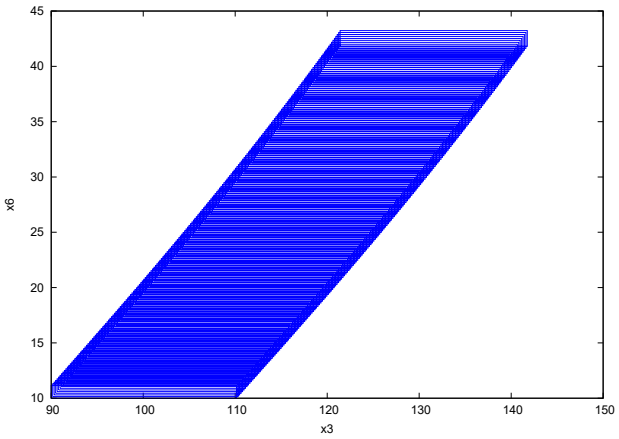

Figure 16: ***ReachNN*.*** Reach sets of the ACC benchmark as obtained from ReachNN*. The x-axis is the lead car position and y-axis is the ego car position. It can be seen from the reachable set that the controller is verified to be safe.

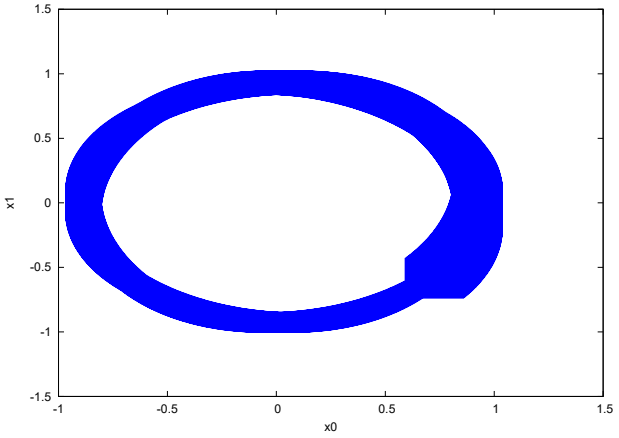

Figure 17: ***ReachNN*.*** Reachability analysis results of benchmark 9. The graph shows reachable set for 6 control steps.

4.3 OVERT

As described earlier, OVERT is designed for use with fundamentally discrete-time systems. This sets it apart from most of the other tools in this survey which are designed for use with continuous-time systems. However in order to evaluate OVERT on benchmarks designed for continuous-time tools, we take the differential equations that are provided in each of these

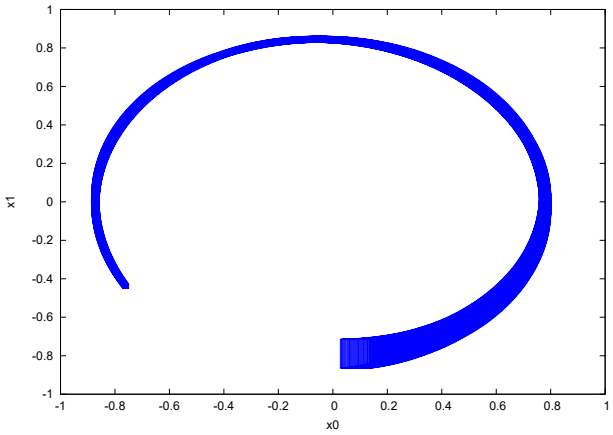

Figure 18: ***ReachNN*.*** Reachability analysis results of Tora-heterogeneous benchmark. The controller is a heterogeneous network with two types of activation functions, ReLU and tanh.

benchmarks, which have the form

$$\dot{\mathbf{x}} = f(\mathbf{x}) \tag{17}$$

and turn them into difference equations using a pre-specified integration scheme. We then obtain discrete-time equations of the form

$$\mathbf{x_{t+1}} = f(\mathbf{x_t}) \tag{18}$$

Consequently, the discrete-time dynamical systems OVERT is evaluated on are different than the continuous-time dynamical systems that most of the other tools are evaluated on, and the results cannot be directly compared. As an example, we look at the TORA benchmark problem. We create difference equations for the TORA problem by using Euler integration with a timestep of 1. We then apply the provided controller to both the discrete time system and an approximation of the continuous time system. The difference between the dynamical systems are illustrated in Figure 19. It is clear from the figure that the same properties will not hold in both the continuous and discrete-time problems.

In the following section, we present results for OVERT on four benchmarks. We present results demonstrating OVERT's capability to solve both satisfiability and reachability problems. For the reachability problem, we find concrete reach sets at specified numbers of timesteps into the future by optimizing the upper and lower bounds of the output variables at those timesteps. We can then intersect these reach sets with a desired target or avoid set to answer satisfiability queries. Alternatively, we can encode the specification into the solver and solve a feasibility problem without explicitly producing the concrete reach set. Using a concrete reachability approach provides intuition about the the evolution of the system, but loosens the overapproximation. The satisfiability approach is faster to run and produces fewer spurious counter examples, but does not produce as intuitive a visualization.

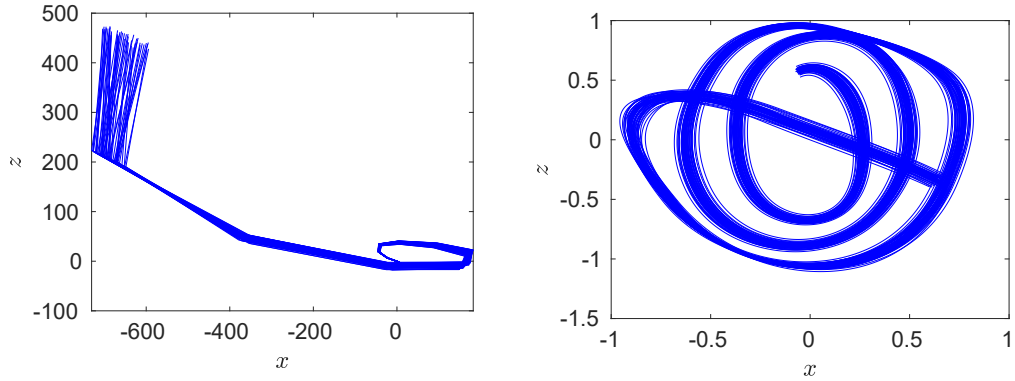

Figure 19: ***OVERT.*** On the left, simulations of a discrete-time version of the TORA problem, using the provided controller. On the right, an approximation of the continuous time system integrated using ode45, with the same provided controller.

4.3.1 Single Pendulum

The results for the single pendulum benchmark can be seen in Figure 20. The discrete-time safety specification $x_1 \in [0, 1]$ is violated at $n_t = 10$. We use the satisfiability formulation to find a counter-example at time step $n_t = 10$ in 70 seconds. We continue checking the specification up to $n_t = 40$ and find that it is not violated for any timesteps n_t between 11 and 40. Checking up to timestep $n_t = 40$ took 250 seconds.

We also explore the evolution of the system using the reachability approach. Figure 21 shows the evolution from time $t = 0$ to time $t = 40$. We confirm that the specification is violated at time step $n_t = 10$, but not at any subsequent time step up to $n_t = 40$. Checking all time steps up to $n_t = 40$ took 10 minutes. The black dashed boxes represents a baseline that concretizes the reach set at every timestep. This incurs unecessary looseness in the overapproximation, and unecessary computational cost. In contrast, the red squares represent concretizing the reach set only four times between timesteps 10 and 40, and are much tighter. The orange dots represent the result of monte carlo simulations of the discrete-time neural network control system at time $n_t = 40$. The red square at time $n_t = 40$ computed using OVERT very closely contains the sampled points.

4.3.2 Sherlock-Benchmark-9: TORA

For the TORA problem, the ODEs are discretized into difference equations using forward Euler integration with a timestep of $\Delta t = 0.1$. The original controller was designed for a continuous system with a control interval of 1 second and was not stable for a forward-Euler-discretized version of the problem with $\Delta t = 1$. Consequently, we chose a smaller Δt in attempt to achieve a more stable closed loop system, but is still not as stable as the continuous version of the system. The decreased stability makes the problem more challenging to verify. The reachability results for a discrete-time version of the TORA benchmark can be seen in Figure 22. The specification of staying within $[-2, 2]^4$ is not violated for the timesteps that we calculate, however, recall that the properties that are true of the continuous system may or may not be true of the

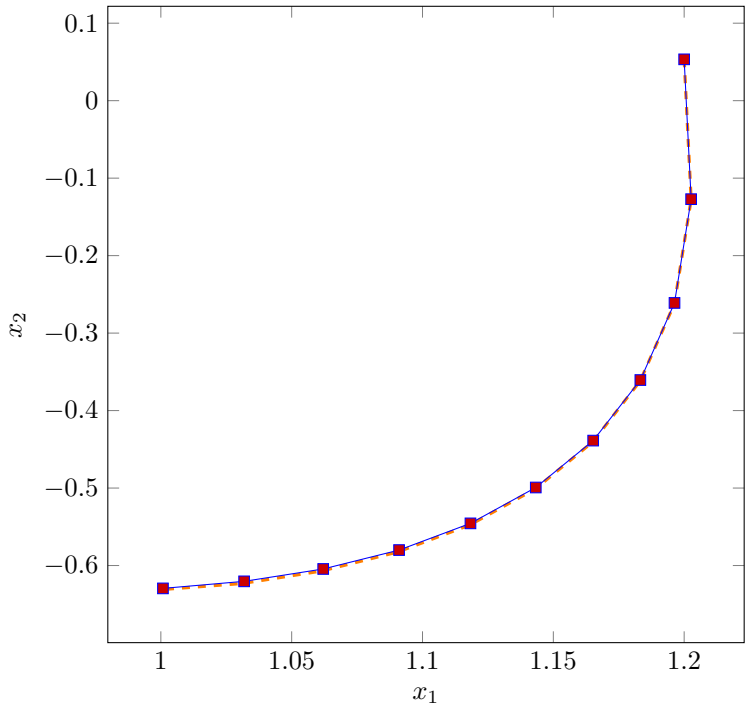

Figure 20: ***OVERT*** satisfiability results on the single pendulum problem. An abstract counter example is found at $n_t = 10$ (blue) and due to the tightness of the overapproximation is easily concretized to a true counter example (red dash)

altered discrete time system. As before, the black boxes represent concretizing the reach set at every timestep, and the red boxes represent concretizing only at select timesteps; in this case timesteps 5,10,13 and 16. Ten hours of computation time were required to produce the sets shown in Figure 22.

4.3.3 Sherlock-Benchmark-10: Car

For the car problem, the ODEs are discretized into difference equations using forward Euler integration with a timestep of $\Delta t = 0.1$. This is reduced from the timestep in the continuous version of the problem, as in the TORA example. This is done in order to achieve increased stability from the controller, as the controller was not trained for the discretized system. Even with this change, the controller is not as stable for the discrete-time version of the system as it is for the continuous-time problem, increasing the difficulty of the verification problem. Figure 23 shows the reachability results for 21 timesteps. The red boxes show where the reach set was concretized at timesteps 6, 11, 16 and 21, and the black boxes demonstrate a baseline approach that concretizes the reach set at every timestep. In this particular example, the reach set is not substantially tighter when only concretizing intermittently. The specification for the continuous time system cannot be evaluated as the discrete-time system is fundamentally a different system. It took 17 hours and 45 minutes to compute the reach sets shown.

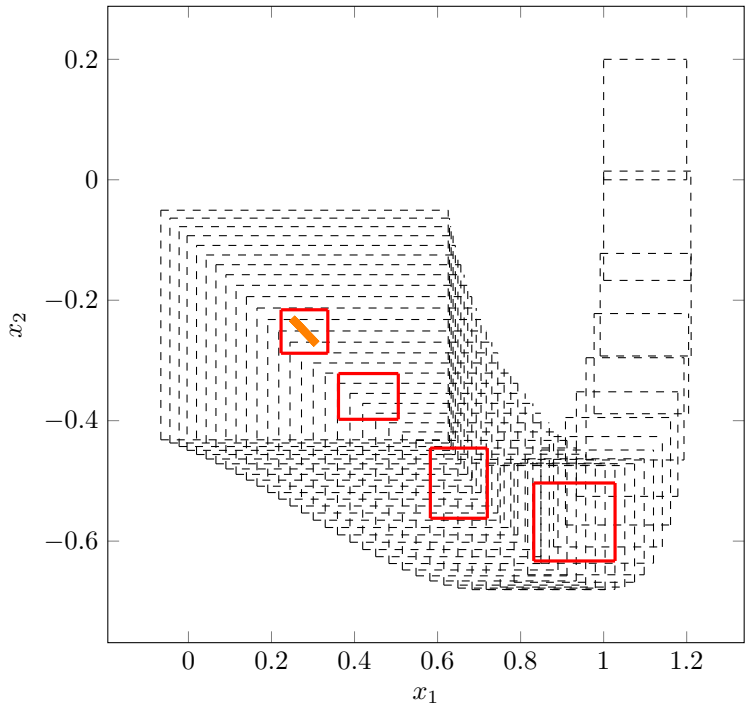

Figure 21: ***OVERT*** reachability results on the single pendulum problem. Red squares represent concretizing only four times over 40 timesteps, and black squares represent a baseline aproach concretizing the reach set at every time step.

4.3.4 ACC

For the Adaptive Cruise Control (ACC) benchmark, the differential equations were discretized using forward Euler integration with a timestep of $\Delta t = 0.1$. The results for the ACC problem can be seen in Figure 24. The reachable set was computed 20 and 35 discrete timesteps into the future using intermittent concretization (shown in red), and is compared to the reachable sets obtained when concretizing at every timestep (show in black). Both red and black reachable sets contain the monte carlo simulations (shown in orange) quite closely. This benchmark required 377 seconds of computation time. We were not able to check the specification of maintaining a safe distance as this was a specification designed for the continuous-time system.

5 VenMAS

The following section reports results obtained using the VenMAS tool. Before delving into the details of each benchmark, we would like to highlight that VenMAS was originally developed to solve verification of discrete-time systems. For this reason, continuous-time benchmarks were discretized using a first order Euler integration scheme.

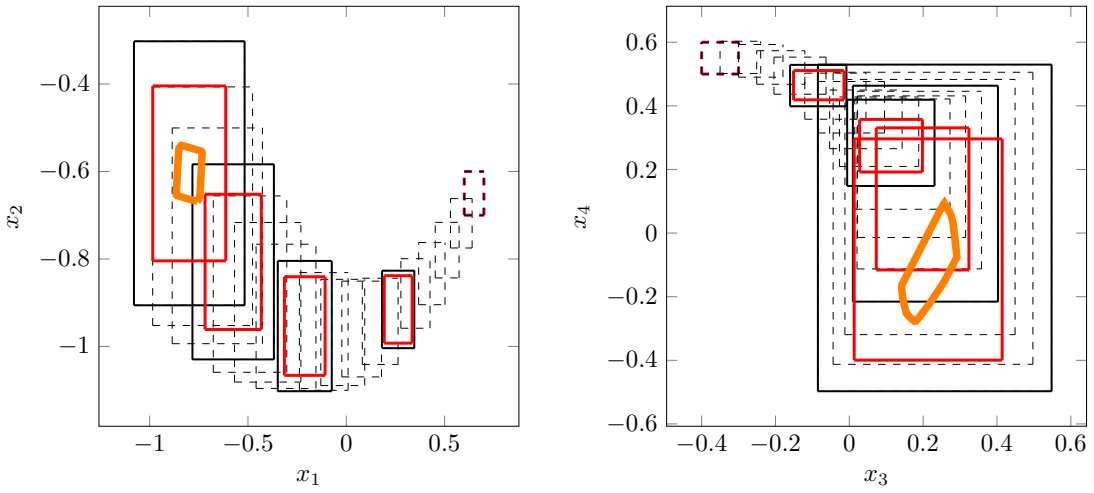

Figure 22: ***OVERT*** reachability results on the TORA problem. The reach set has been concretized at timesteps $n_t = 5, 10, 13, 16$.

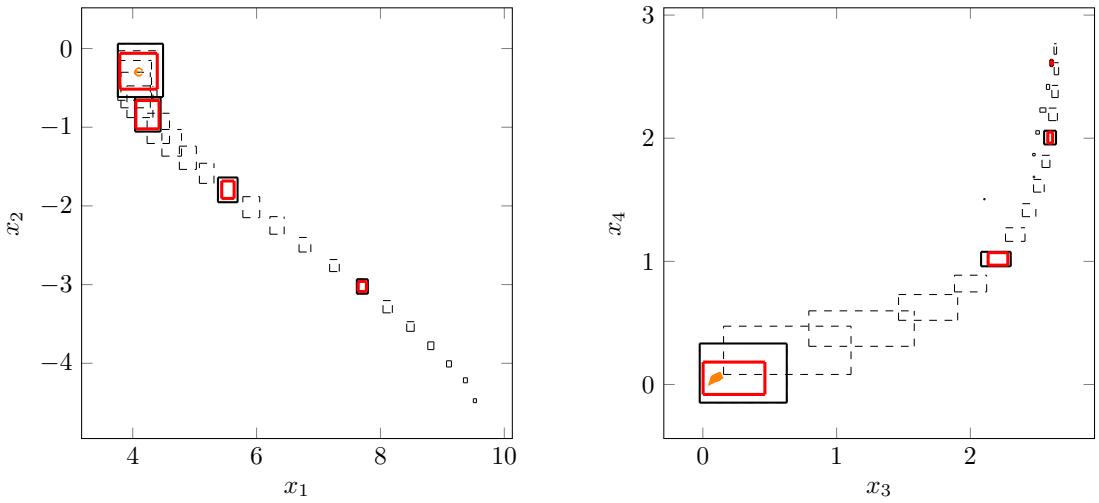

Figure 23: ***OVERT*** reachability results on the Sherlock-Bencharmark-10 Car problem, concretized at timesteps $n_t = 6, 11, 16, 21$.

We also recall that VenMAS is a sound and complete (modulo numerical issues) tool for verification of closed-loop systems with ReLU-based neural network controllers that solves the verification problem via a reduction to the feasibility problem of mixed-integer linear programs (MILPs). Whenever the specification is satisfied, the generated program is infeasible, while when the specification is violated, the program is feasible and a counter-example demonstrating

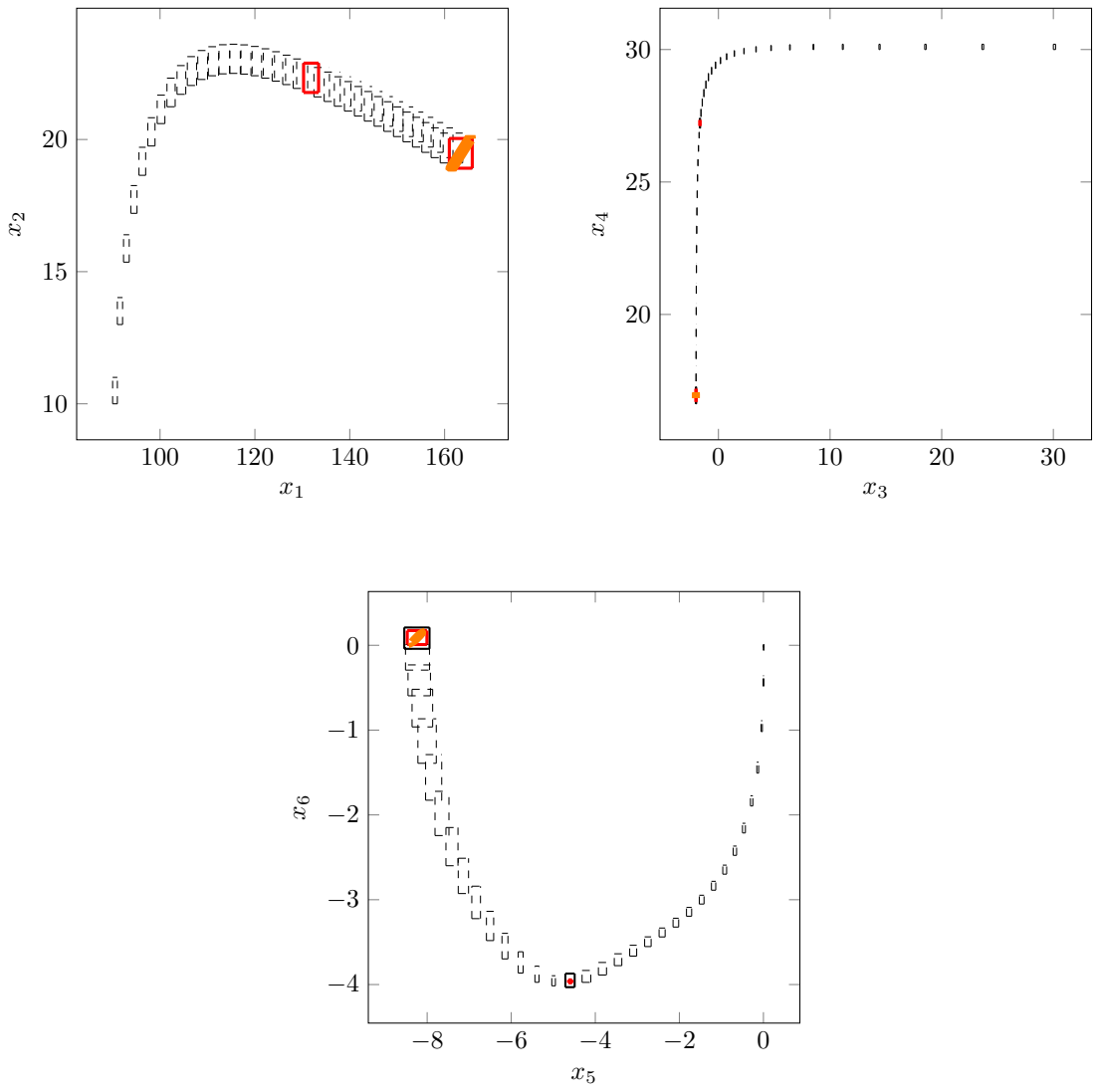

Figure 24: ***OVERT*** reachability results on the ACC problem. Black boxes are reach sets computed if using concretization at every timestep. Red boxes are obtained by concretizing only at $n_t = 20, 35$.

violation of the property can be extracted from the model of the program. Therefore, VenMAS does not compute flow pipes or reachable sets. Additionally to that, non-linear terms that appear in some benchmarks could not be handled directly and had to be approximated – neural approximators were used for this purpose.

5.1 Airplane

There are no results for this benchmark due to the highly non-linear nature of the benchmark.

5.2 ACC

VenMAS was able to prove that there is no collision in the following 5 seconds (i.e., 50 time steps) starting from the set of initial states. In total, it took VenMAS 28.84 seconds to show that. The neural network controller considered was that one with 5 layers (of 20 nodes each).

5.3 Double pendulum

There are no results for this benchmark due to the highly non-linear nature of the benchmark.

5.4 Tora (Sherlock-Benchmark-9)

VenMAS was able to prove that the system stays within the box $[-2, 2]^4$ for a time window of 20 seconds (with control period of 1 second) starting from the set of initial states. It took VenMAS a total of 34.85 seconds to do so. To approximate the sine function, we trained a neural network with 2 hidden layers of 16 and 8 nodes, respectively.

5.5 Unicycle (Sherlock-Benchmark-10)

VenMAS was able to show that the system does indeed reach the required box after 24 steps from all the initial states (taking 37.35 seconds to prove reachability for 24 steps). Also, VenMAS has shown that up to at least 15 time steps, there are initial states from which the desired box is not reached (taking 975.81 seconds to disprove reachability for 15 steps). For time steps from 16 to 23, VenMAS was not able to conclude anything within the time limit of $1,800$ seconds. We conjecture in such cases Gurobi runs in a worst-case behavior where usual solving heuristics are not effective and as a result the solution space must be searched exhaustively. To approximate the non-linearities $x_4 sin(x_3)$ and $x_4 cos(x_3)$, we trained two neural networks with 3 hidden layers of 12 nodes each.

5.6 Single pendulum

VenMAS was able to prove that the systems is not safe after 10 time steps (sampling time 0.05 seconds) taking 0.27 seconds. Additionally, VenMAS has shown that after 11 time steps the system does converge to the safe region and stays there up until 20 time steps, taking a total time of 383 seconds. To approximate the sine function, we trained a neural network with 2 hidden layers of 16 and 8 nodes, respectively.

5.7 Tora heterogeneous

There are currently no results for this benchmark as VenMAS only supports ReLU activations.

5.8 VCAS

VenMAS has shown that

- for initial climbrate of -19.5, the ownship manages to stay out of the NMAC zone for all 10 time steps, taking a total time of 1.62 seconds.

- for initial climbrate of -22.5, the ownship enters the NMAC zone for one time step (namely, after 3 steps) and then leaves it for all the remaining time steps, taking a total time of 1.30 seconds. The altitude h in a trace witnessing a counter-example for 3 time steps is as follows: $h(0) = -129.0$, $h(1) = -110.525$, $h(2) = -100.10$ and $h(3) = -97.725$.

- for initial climbrate of -25.5, the ownship enters the NMAC zone for 3 time steps (2, 3 and 4) and then leaves it for all the remaining time steps, taking a total time of 1.87 seconds. The altitude h in a trace witnessing a counter-example for 4 time steps is as follows: $h(0) = -133.0$, $h(1) = -111.525$, $h(2) = -98.10$, $h(3) = -93.395$ and $h(4) = -97.4125$.

- for initial climbrate of -28.5, the ownship enters the NMAC zone for 4 time steps (2, 3, 4 and 5) and then leaves it for all the remaining time steps, taking a total time of 4.24 seconds. The altitude h in a trace witnessing a counter-example for 5 time steps is as follows: $h(0) = -133.0$, $h(1) = -109.195$, $h(2) = -94.1125$, $h(3) = -87.08$, $h(4) = -88.76$ and $h(5) = -99.175$.

When checking the specifications, VenMAS is able to take into account all possible choices of the acceleration by the pilot when producing the MILP encoding, thus showing true safety or finding a counter-example when it exists.

6 Category Status and Challenges

In the second iteration of the AINNCS category at ARCH-COMP, the participating tools NNV, OVERT, ReachNN* and VenMAS successfully analyzed different aspects of the benchmark problems. In spite of some success analyzing the benchmarks, the primary outcome of this second iteration of the AINNCS category are the challenges that arose in the competition. We discuss these challenges next.

Hybrid Controllers: Some controllers involve a hybrid nature. This type of controller only appears in the VCAS benchmark. This is a very complex control system formed by 9 different neural networks that are chosen based on plant's states. These controllers have also a bang-bang output characteristic, meaning that the output range is not continuous, but is chosen from a discrete set of values depending on the current neural network executed, as well as all output values and the aircraft states. We observe that only half of the tools have results for this benchmark, although in the case of NNV, it is noted that it required extra work to add the support for this benchmark, and it remains a challenge to generalize these type of systems.

Plant Models: This year we have only considered nonlinear plants, both in discrete and continuous time. A majority of the tools only support discrete or continuous time, with NNV being the only tool with support for both type of dynamics. We plan to add linear as well as hybrid automata plants in future iterations, as we look to report a more complete analysis of the participating verification tools. Hybrid automata plants will be especially interesting with the complex nature of combined continuous and discrete dynamics, which is very challenging for current AINNCS verification tools.

Activation Function Types: 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. This is a step forward from last year's competition, as nonlinear activation functions were included. However, not all tools have support for all activation functions, with some supporting only ReLU and linear activation functions at the time the report was written.

Neural Network Architectures and Parameterization: When we compare the neural network architectures presented in this work with some of the networks that can be analyzed in absence of the plant, these are fairly simple, in the sense none of the networks have more than a thousand neurons, and none exceed 5 hidden layers in their architecture. Also, the maximum number of inputs and outputs of the controllers are 12 and 6, respectively, in the airplane benchmark. If we consider the VCAS benchmark, these networks have 9 outputs, although these are translated into a single input to the plant model. Thus, the dimensionality of the neural network controllers and plant states have significantly increased compared to last year. Additionally, the verification results presented assumed the neural networks are fixed, while other parameterizations are possible, some of which were partly explored (e.g., using different network architectures or activation functions for a given plant). In any event, there are state-space explosion and scalability issues to address in both the neural network controllers and plant analysis.

Time horizons: Similar to last year's competition, all the tools performed bounded (finite) time horizon verification analysis, also known as bounded model checking, where the main difference is that there are two participating tools that do not rely on reachability analysis methods to analyze safety. This was accomplished up to some pre-specified number of control periods k with some pre-specified sampling period for the continuous dynamics' evolution. Alternative approaches for performing unbounded (infinite) time horizon verification exist, such as those building on barrier certificates, a form of continuous analog of the classical inductive invariance proof rule. The existing methods could incorporate invariance checks on the computed reachable states to attempt to determine if the reachability analysis reaches a fixed-point (if the reachability analysis terminates, which for the class of systems considered, is not guaranteed as the reachability analysis with nonlinear plants is undecidable). However, no current methods evaluated in the competition utilize this approach, and this is a promising avenue for future work to provide guarantees beyond finite time horizons.

Model Formats: Following previous discussions, we removed the Simulink simulations and the plants' SpaceEx models from the competition. We have found 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[3], .mat format[4], and the original format used by proposer of the benchmark. ONNX format was very convenient as most of the participating tools have integrated ONNX into their frameworks this year. However, we found that there are discrepancies among the

[3]Open Neural Network Exchange: https://github.com/onnx/onnx
[4]Direct input format used by *NNV* without transformation.

different versions and frameworks these ONNX models were created from. Having a unified ONNX version remains a challenge, but we are closer to achieving this goal, and initiatives more focused on neural network verification, such as VNN-LIB[5] and VNN-COMP[6], may help toward this goal.

7 Conclusion and Outlook

This report presents the results on the second ARCH friendly competition for closed-loop systems with neural network controllers. In the second edition of this category, four tools were applied to attempt to solve 7 different benchmark problems, namely NNV, OVERT, ReachNN*, and VenMAS. 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. We note that each tool has unique strengths and that not all of the specificities can be highlighted within a single report. However 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. We observe that the numbers of benchmarks and difficulty of these have increased from the first iteration, which is a good indicator for this growing and maturing field.

We would also like to encourage other tool developers to consider participating next year. All authors agree that although the participation consumes time, we have gained unique insights that will allow us to improve in the next iteration. Particularly those items listed in the status and challenges section. The reports of other categories can be found in the proceedings and on the ARCH website: cps-vo.org/group/ARCH.

8 Acknowledgments

The material presented in this report is based upon work supported by the National Science Foundation (NSF) under grant number FMitF 1918450, the Air Force Office of Scientific Research (AFOSR) through contract number FA9550-18-1-0122, and the Defense Advanced Research Projects Agency (DARPA) Assured Autonomy program through contract number FA8750-18-C-0089. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFOSR, DARPA, or NSF.

A Specification of Used Machines

A.1 M_{nnv}

- Processor: Intel Core i7-8750H CPU @ 2.20GHz x 12
- Memory: 32 GB

[5] http://www.vnnlib.org/
[6] https://github.com/verivital/vnn-comp/

A.2 $\mathbf{M_{Overt}}$

- Processor: Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
- RAM: 64 GB

A.3 $\mathbf{M_{ReachNN*}}$

- 12-core 3.60 GHz Intel Core i7
- Memory: 32 GB
- GPU: NVIDIA GeForce RTX 2060 with 6 GB RAM

A.4 $\mathbf{M_{VenMAS}}$

- Processor: Intel Core i7-7700K CPU @ 4.20GHz
- Memory: 16 GB

References

[1] *Proceedings of the 7th International Conference On Formal Methods In Software Engineering, FormaliSE 2019, collocated with ICSE 2019, Montréal, Canada, May 27, 2019*. ACM, 2019.

[2] 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.

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

[4] 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.

[5] 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.

[6] Rajeev Alur. Formal Verification of Hybrid Systems. In *Proceedings of the Ninth ACM International Conference on Embedded Software*, EMSOFT '11, pages 273–278, New York, NY, USA, 2011. ACM.

[7] 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.

[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] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification*, pages 258–263. Springer, 2013.

[10] 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.

[11] 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.

[12] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A. Mann, and Pushmeet Kohli. A Dual Approach to Scalable Verification of Deep Networks. *CoRR*, abs/1803.06567, 2018.

[13] Jiameng Fan, Chao Huang, Wenchao Li, Xin Chen, and Qi Zhu. Towards verification-aware knowledge distillation for neural-network controlled systems. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2019.

[14] 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.

[15] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2020.

[16] Chao Huang, Jiameng Fan, Wenchao Li, Xin Chen, and Qi Zhu. Reachnn: Reachability analysis of neural-network controlled systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–22, 2019.

[17] 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.

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

[19] 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.

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

[21] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, and Fuad E. Alsaadi. A Survey of Deep Neural Network Architectures and their Applications. *Neurocomputing*, 234:11 – 26, 2017.

[22] 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.

[23] 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.

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

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] Weiming Xiang and Taylor T. Johnson. Reachability analysis and safety verification for neural network control systems. *CoRR*, abs/1805.09944, 2018.

[32] 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.

[33] Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. Output reachable set estimation and verification for multi-layer neural networks. *CoRR*, abs/1708.03322, 2017.

[34] Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. Reachable set computation and safety verification for neural networks with relu activations. *CoRR*, abs/1712.08163, 2017.

[35] Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. Output reachable set estimation and verification for multi-layer neural networks. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, 2018.

[36] Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. Specification-guided safety verification for feedforward neural networks. *CoRR*, abs/1812.06161, 2018.

[37] Weiming Xiang, Hoang-Dung Tran, Xiaodong Yang, and Taylor T. Johnson. Reachable set estimation for neural network control systems: A simulation-guided approach. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–10, 2020.