Safety Verification of Neural Network Controlled Systems

Arthur Clavière, Eric Asselin, Christophe Garion, Claire Pagetti

Safety Verification of Neural Network Controlled Systems

Arthur Clavière[1], Eric Asselin[1], Christophe Garion[2] and Claire Pagetti[3]

[1]Collins Aerospace, France [2]ISAE-SUPAERO, France [3]ONERA, France

Abstract

In this paper, we propose a system-level approach for verifying the safety of neural network controlled systems, combining a continuous-time physical system with a discrete-time neural network based controller. We assume a generic model for the controller that can capture both simple and complex behaviours involving neural networks. Based on this model, we perform a reachability analysis that soundly approximates the reachable states of the overall system, allowing to achieve a formal proof of safety. To this end, we leverage both validated simulation to approximate the behaviour of the physical system and abstract interpretation to approximate the behaviour of the controller. We evaluate the applicability of our approach using a real-world use case. Moreover, we show that our approach can provide valuable information when the system cannot be proved totally safe.

1 Introduction

Recently, feedforward deep neural networks have been successfully used for controlling physical systems, such as self-driving cars [17, 5, 28] and unmanned aerial vehicles [16]. The combination of a physical system with a neural network based controller is sometimes known as a *neural network controlled system*. If such a system is considered as *safety-critical*, meaning that a failure of the system could have serious consequences, then a particular effort needs to be made to demonstrate its safety. More precisely, one has to show evidence that the system fulfills a set of safety requirements, such as, in aeronautics, *"A catastrophic failure shall occur with a probability less than* 10^{-9} *per hour of flight"*.

Usually, to achieve this objective, the system has to be developped in accordance with stringent standards *e.g.*, ED-79A/ARP-4754A [1] in aeronautics. Such standards require several analyses to be performed, including safety assessment with fault trees. Moreover, together with these analyses, the system requirements have to be refined at the *item level*, with the aim of achieving a *correct*, *comprehensive* specification for each item composing the system. Then, the development of each item must be performed in compliance with dedicated standards. For example, in aeronautics, the ED-12C/DO-178C [8] standard prescribes several verification activities to prove that a software item behaves *exactly* as expected.

However, this classical approach is not applicable to neural network controlled systems. The reason for this is two fold. First, one cannot refine the system requirements at the neural network level. Indeed, most of the time, one cannot achieve a correct, comprehensive specification for the expected behaviour of a neural network. Generally, the expected behaviour of a network consists of a set of example data, which is a *pointwise, non-comprehensive* specification. Secondly, existing standards such as ED-12C/DO-178C are not applicable to the development of neural network items. In particular, provided a comprehensive specification for the expected behaviour of a neural network can be defined, the learning process does not guarantee the correcteness of the resulting network. As a consequence, verifying that a network behaves exactly as expected may be infeasible, precisely because it does not.

To tackle these issues, we propose an alternative approach for demonstrating the safety of a neural network controlled system. This alternative approach aims at providing evidence that the overall system is safe, without performing item-level refinements and analyses. To this end, we leverage a model of the overall system, which accurately represents the items and their interactions. Then, a reachability analysis is performed on this model, with the aim of demonstrating that no reachable state can lead to a failure of the system.

The contributions of this paper are: (1) the definition of a *realistic model* that can capture complex, real-world neural network controlled systems, involving one or more ReLU networks trained with supervised learning together with a pre-processing and a post-processing, (2) a reachability-based approach that allows to *formally verify* the absence of errors leading to a failure of the system, and (3) an evaluation of the *applicability* of our approach using a real-world use case.

The paper is organized as follows. Section 3 introduces the ACAS Xu use case, a real-world neural network controlled system that illustrates the applicability of our approach. Section 4 describes our model of a neural network controlled system and section 5 defines the safety verification problem that we address. Section 6 details our reachability analysis for solving the verification problem and section 7 presents the experimental results on the ACAS Xu use case.

2 Related work

Neural network level In the past few years, some progress has been made towards a more comprehensive specification for the expected behaviour of a neural network. Indeed, several research works have identified *local* expected behaviours contributing to the overall expected behaviour of the network. Typically, a local behaviour consists of a pre-condition about the input of the network together with a post-condition about its output. An example of such a property is *adversarial robustness* (also called local robustness) which captures the capability of the network to react correctly to a slight perturbation of a *given* input [12, 26]. In recent years, there has been significant interest in verifying neural networks against this type of property, which has been shown to be a NP-complete problem [12]. Several dedicated *formal methods* have been proposed, with the advantage of providing a *sound* analysis, meaning that the network is said correct only if it is actually correct. Some of these specialized formal methods are based on Satisfiability Modulo Theory solving [12, 19], with the advantage of providing a *complete* analysis *i.e.*, the network is said incorrect only if it is actually incorrect. However, these methods are often expensive for large, real-world sized networks. In order to offer a more *scalable* analysis, other dedicated formal methods have been proposed, relying on abstract intepretation to soundly *approximate* the semantics of the network [25, 13, 23, 24]. Yet, as they consist of an over-approximation, these methods do not provide a complete analysis.

Our work does not address the safety objectives at the neural network level, so we do not seek to identify new local properties or to improve the existing verification techniques. However, we aim at using abstract interpretation based techniques to analyze the behaviour of the overall system. Indeed, such methods scale well to large networks and they provide not only a yes-or-no answer to a verification problem but also an approximation of the network semantics, that is helpful when reasoning about the overall system.

System level Verifying the safety requirements at the system level, which corresponds to our approach, has been the object of a lot of insightful research. Indeed, there has been significant interest in verifying the safety of *hybrid systems*, exhibiting both continuous-time and discrete-time dynamics *e.g.*, a physical system combined with a discrete-time controller. Among the proposed methods, *falsification* aims at finding trajectories that violate a given safety property [4, 27]. Yet, even though falsification can prove that the system is unsafe, it cannot prove that the system is safe. *Reachability analysis* can provide such a proof of safety by constructing a *sound* approximation of the reachable states of the system and demonstrating that no reachable state can lead to a failure [6, 2, 21]. However, the classical reachability methods are not directly applicable to neural network controlled systems, due to the hardness of characterizing the input-output mapping of a neural network. Very recently, in the same vein as this paper, some research works have addressed the problem of verifying the safety of neural network controlled systems [20, 9, 14, 11]. These works all assume a physical system combined with a periodically-scheduled controller that is a *single* neural network *i.e.*, the input of the network is the sampled state of the physical system and the output of the network is the actuation command. To ensure the safety of such a system, they propose dedicated methods, all relying on reachability analysis.

However, these methods are not applicable to *complex* neural network controlled systems such as the ACAS Xu. Indeed, the controller used in the ACAS Xu is more sophisticated than a single neural network and the methods cited above cannot handle such a controller. [7] has proposed an *ad hoc* reachability approach for verifying the safety of the ACAS Xu at the system level, but the proposed method is not totally sound as it does not evaluate the reachable states for all instants

but only for a set of *discrete* instants. Moreover, it computes the reachable states by exploring the entire state space even though not all states are reachable. We propose here a *generic* approach for *soundly* verifying the safety of complex systems like the ACAS Xu, by exploring only the reachable states.

3 Use case

The safe integration of Unmanned Aerial Vehicles (UAVs) into the air traffic requires them to have collision avoidance capabilities. For this purpose, the standardization group RTCA SC 147 [22] has recently developped a dedicated controller, namely the Airborne Collision Avoidance System for Unmanned Aircraft (ACAS Xu). The role of ACAS Xu is to avoid any collision between the *ownship*, equipped with the controller, and an encountered aircraft called the *intruder*, equipped or not with the controller. To this end, the ACAS Xu periodically provides the ownship with a *horizontal maneuver advisory*, being either clear-of-conflict (COC), weak left turn (WL), weak right turn (WR), strong left turn (SL) or strong right turn (SR). The optimal advisory is extracted from a set of lookup tables, depending on the previous advisory and six variables describing the encounter between the two aircraft, defined in Fig. 1: (1) the distance ρ from ownship to intruder, (2) the angle θ to intruder relative to ownship heading direction, (3) the heading angle ψ of intruder relative to ownship heading direction, (4) the velocity v_{own} of ownship, (5) the velocity v_{int} of intruder and (6) the time t_{sep} until loss of vertical separation. These six variables are computed from the input signals from the transponder and the sensors of the ownship *e.g.*, air-to-air radar, electro-optics/infrared sensors, cameras. The main weakness of the ACAS Xu controller is the associated storage requirements, over 2GB, which is too large for legacy avionics [16].

Figure 1: The 2D geometry of the encounter between the two aircraft (left) and the (illustrative) reachable trajectories of intruder relative to ownship from a subset $\mathbf{I}'$ of the possible initial states, with $\mathbf{E}$ representing a collision cylinder around the ownship and $\mathcal{R}$ delimiting the range of the ownship sensors (right).

Recently, an alternative design for the ACAS Xu controller has been proposed, with dramatically reduced memory footprint (about 3 MB). It consists of a collection of 45 neural networks approximating the lookup tables. Each single network approximates a table corresponding to a fixed previous advisory and a given interval for t_{sep}. As the possible values for t_{sep} have been divided into 9 intervals and 5 possible advisories exist, the resulting controller uses 45 networks. In addition to improving storage efficiency, this novel design also offers reduced runtime together with better performances, alerting the ownship earlier [16]. However, due to the complexity of the neural networks composing the controller, we lack a proof that no collision can happen, whatever the initial state of the two aircraft (see Fig. 1).

This use case will serve as an illustration of our approach in the rest of this paper. Our goal is to show evidence that the controller is effectively safe *i.e.*, it does prevent near mid-air collision.

4 System model

4.1 Closed-loop system

We assume a *closed-loop system* $\mathcal{C}$ that is the combination of a plant $\mathcal{P}$ and a neural network based controller $\mathcal{N}$. The plant $\mathcal{P}$ is a *continuous-time* system while the controller $\mathcal{N}$ is a *discrete-time* system, executed periodically with period T. They interact by means of a signal sampler and a zero-order-hold. More precisely:

- The state of the plant $\mathcal{P}$ at instant $t \in \mathbb{R}$ is the real-numbered vector $\mathbf{s}(t) \in \mathbb{R}^l$. The evolution of $\mathbf{s}(t)$ is continuous with t and it depends, inter alia, on the actuation command from the controller, denoted by $\mathbf{u}(t) \in \mathbb{R}^d$.
- The j^{th} execution of the controller (or control step) occurs in the time interval $[jT, (j+1)T[$. It takes as input the sampled state $\mathbf{s}_j = \mathbf{s}(jT)$ and it yields the command $\mathbf{u}_{j+1}$ to be applied for next period *i.e.*, $\mathbf{u}(t) = \mathbf{u}_{j+1}\ \forall t \in [(j+1)T, (j+2)T[$. This command $\mathbf{u}_{j+1}$ is taken from a *finite* set $\mathbf{U} = \left\{\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(P)}\right\} \subset \mathbb{R}^d$, representing the possible actuation commands. It is worth noting that the controller is not assumed to execute instantaneously. Its execution time only has to be less than T, as for real systems.

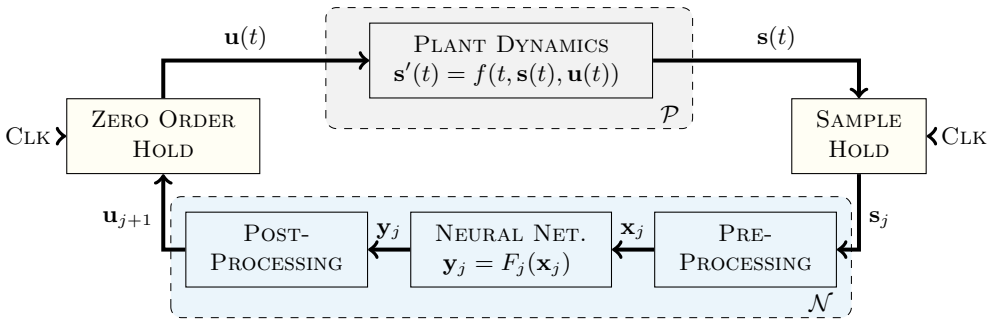


Figure 2: Block diagram of a neural network closed-loop system $\mathcal{C} = (\mathcal{P}, \mathcal{N})$.

Overall, the state of the closed-loop system $\mathcal{C}$ is the 2-tuple $\phi(t) = (\mathbf{s}(t), \mathbf{u}(t))$ and we denote by $\phi_0 = (\mathbf{s}_0, \mathbf{u}_0) \in \mathbf{I}$ the initial state of $\mathcal{C}$, wherein $\mathbf{I} \subseteq \mathbb{R}^l \times \mathbf{U}$ is the set of the possible initial states. Moreover, we consider a set of *erroneous* states $\mathbf{E} \subset \mathbb{R}^l \times \mathbf{U}$ such that a state $\phi(t) \in \mathbf{E}$ causes a potentially catastrophic failure of $\mathcal{C}$. It is thus expected that $\mathcal{C}$ does not reach a state in $\mathbf{E}$. We also assume that $\mathcal{C}$ terminates when its state $\phi(t)$ belongs to a set $\mathbf{T} \subset \mathbb{R}^l \times \mathbf{U}$, with $\mathbf{T} \cap \mathbf{E} = \emptyset$ to ensure a safe behaviour. Here, $\mathbf{T}$ can be seen as a set of *target* states, corresponding to $\mathcal{C}$ having successfully achieved its mission. It is thus expected that $\mathcal{C}$ terminates in a *finite* amount of time, whatever the initial state. We denote by $\tau \in \mathbb{R}$ the expected (or estimated) upper bound on this amount of time, independently of the initial state. Additionally, we set by definition $\phi(t) = \perp$ after the termination of the closed-loop system $\mathcal{C}$ *i.e.*, if $t_{\text{end}} \leq \tau$ satisfies $\forall t < t_{\text{end}},\ \phi(t) \notin \mathbf{T}$ and $\phi(t_{\text{end}}) \in \mathbf{T}$ then $\phi(t) = \perp\ \forall t \in]t_{\text{end}}, \tau]$. In other words, the bottom element symbolically represents the "terminated" state of $\mathcal{C}$.

Finally, as the combination of a deterministic plant $\mathcal{P}$ and a deterministic controller $\mathcal{N}$ (see sections 4.2 and 4.3), the closed-loop system $\mathcal{C}$ has a deterministic behaviour. More precisely, for a given initial state $\phi_0 \in \mathbf{I}$, there exists a unique function $\phi_{\phi_0} : [0, \tau] \to \mathbb{R}^l \times \mathbf{U} \cup \{\perp\}$ such that $\phi_{\phi_0}(t)$ is the state of $\mathcal{C}$ at instant $t \leq \tau$. This hypothesis is important for properly defining the verification problem that we address, as well as demonstrating the soundness of our procedure.

Example 1 *In the case of the ACAS Xu controller, we consider the plant $\mathcal{P}$ that is composed of both the ownship and the intruder. For simplicity, we assume that the two aircraft are at the same altitude, meaning that t_{sep} equals 0. Consequently, we define the state of $\mathcal{P}$ at instant t as the real-numbered vector $\mathbf{s}(t) = (x(t)\ y(t)\ \psi(t)\ v_{own}(t)\ v_{int}(t))^T$ where $x(t), y(t)$ are the 2D cartesian coordinates of intruder relative to ownship, $\psi(t)$ is the heading angle of intruder relative to ownship heading direction (measured counter clockwise), $v_{own}(t)$ and $v_{int}(t)$ denote the velocities of ownship and intruder respectively (see Fig. 3). The neural network based controller $\mathcal{N}$ has a period $T = 1s$. It outputs the actuation command $u(t) \in \mathbb{R}$ that is the turn rate of ownship, measured counter clockwise. This command is taken from the set $\mathbf{U} = \{0\ deg/s, 1.5\ deg/s, -1.5\ deg/s, 3\ deg/s, -3\ deg/s\}$,*

of which values represent COC, WL, WR, SL and SR respectively. Overall, an initial state $\phi_0 = (\mathbf{s}_0, u_0)$ *of the closed-loop* $\mathcal{C}$ *corresponds to the intruder being detected by ownship for the first time. Therefore, the initial position* (x_0, y_0) *of intruder lies along a circle* $\mathcal{R}$ *centered on ownship and with a radius* r *equal to the range of the ownship sensors (see Fig. 1). Here we consider that* $r = 8000$ *ft, which is a reasonable hypothesis. Furthermore, the initial angle* ψ_0 *is such that the intruder penetrates the circle* $\mathcal{R}$ *i.e.* ψ_0 *belongs to a cone delimited by the tangent to* $\mathcal{R}$ *at the point* (x_0, y_0)*. The initial actuation command* u_0 *is* 0.0 *deg/s, corresponding to a Clear-of-Conflict, and we assume for simplicity that* $v_{own,0} = 700.0$ *ft/s and* $v_{int,0} = 600.0$ *ft/s. The set of the possible initial states* $\mathbf{I}$ *is thus defined by the set of the possible tuples* (x_0, y_0, ψ_0)*. Additionally, we consider a set* $\mathbf{E}$ *of erroneous states representing a collision between the two aircraft. Such a collision happens when the intruder enters the collision circle around ownship, with a radius of* 500 *ft [18], hence* $\mathbf{E} = \{\phi(t) = (\mathbf{s}(t), u(t)) \in \mathbb{R}^l \times \mathbf{U} \mid \sqrt{x(t)^2 + y(t)^2} < 500.0 \text{ ft}\}$*. Finally, the closed-loop system terminates when the intruder leaves the circle* $\mathcal{R}$ *i.e. the ownship does not see it anymore:* $\mathbf{T} = \{\phi(t) = (\mathbf{s}(t), u(t)) \in \mathbb{R}^l \times \mathbf{U} \mid \sqrt{x(t)^2 + y(t)^2} > r\}$*. As the two aircraft have different velocities, it is expected that* $\mathcal{C}$ *terminates in a finite amount of time. We take* $\tau = 20s$ *as the upper bound on this amount of time, which is relevant given the values of* v_{own}*,* v_{int} *and* r*.*

4.2 Plant dynamics

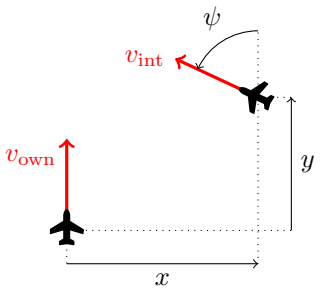
The dynamics of the plant $\mathcal{P}$ *i.e.*, the temporal evolution of its state $\mathbf{s}(t)$, is modelled by an *ordinary differential equation.*

Definition 1 *An ordinary differential equation (ODE) is a relation between a function* $\mathbf{z} : \mathbb{R} \to \mathbb{R}^l$*,* $t \mapsto \mathbf{z}(t)$ *and its derivative* $\mathbf{z}' = \frac{d\mathbf{z}}{dt}$ *of the form* $\mathbf{z}'(t) = f(t, \mathbf{z}(t))$ *wherein* $f : \mathbb{R} \times \mathbb{R}^l \to \mathbb{R}^l$*.*

To take account of the command signal $\mathbf{u}(t)$, the dynamics of $\mathcal{P}$ is of the form $\mathbf{s}'(t) = f(t, \mathbf{s}(t), \mathbf{u}(t))$ wherein $f : \mathbb{R} \times \mathbb{R}^l \times \mathbb{R}^d \to \mathbb{R}$ is assumed to be continuous in t and $\mathbf{u}$ and uniformly Lipschitz continuous in $\mathbf{s}$ *i.e.*, its slope *w.r.t.* $\mathbf{s}$ is uniformly bounded on $\mathbb{R} \times \mathbb{R}^l \times \mathbb{R}^d$. Indeed, under these hypotheses and when $\mathbf{u}(t)$ is a piecewise constant function (as in the case of $\mathcal{C}$), then $\mathcal{P}$ has a deterministic behaviour. More precisely, let us consider a time interval $[0, qT]$ with $q \in \mathbb{N}$ and a given command signal $\mathbf{u}(t)$, constant on $[jT, (j+1)T[$ for $j < q$. There exists a unique function $\mathbf{s}^*$ defined on $[0, qT]$, continuous on $[0, qT]$, such that it verifies the ODE on each open interval $]jT, (j+1)T[$ for $j < q$, and the initial condition $\mathbf{s}(0) = \mathbf{s}_0$.

Proof 1 *The function* $\mathbf{s}^*$ *can be constructed iteratively. The initial condition imposes* $\mathbf{s}^*(0) = \mathbf{s}_0$*. Then, for* $j \in [\![0, q-1]\!]$*, the Picard-Lindelöf theorem ensures the existence and uniqueness of a function* $\mathbf{s}_j^*$ *satisfying* $\mathbf{s}'(t) = f(t, \mathbf{s}(t), \mathbf{u}(jT))$ *and* $\mathbf{s}(jT) = s^*(jT)$*. In order for* $\mathbf{s}^*$ *to be continuous at instants* $t = jT$ *and* $t = (j+1)T$ *and to satisfy the ODE on* $]jT, (j+1)T[$*, it must be such that* $\mathbf{s}^*(t) = \mathbf{s}_j^*(t)\ \forall\ t \in]jT, (j+1)T]$*. Hence the existence and uniqueness of* $\mathbf{s}^*$*.* □

Remark 1 *The function* $\mathbf{s}^*$ *is not derivable at instants* $t = T, 2T, \ldots, (q-1)T$*. This is not quite realistic from a physical point of view as it means that the plant* $\mathcal{P}$ *reacts instantaneously to a new actuation command. However, this is a common hypothesis when modelling such a system.*



$$\begin{cases} x'(t) = -v_{\text{int}}(t) \cdot \sin(\psi(t)) \\ y'(t) = v_{\text{int}}(t) \cdot \cos(\psi(t)) - v_{\text{own}}(t) \\ \psi'(t) = -u(t) \\ v'_{\text{own}}(t) = 0 \\ v'_{\text{int}}(t) = 0 \end{cases} \tag{1}$$

Figure 3: The 2D kinematic model of the plant $\mathcal{P}$ of the ACAS Xu use case, composed of both the ownship and the intruder.

Example 2 *For the ACAS Xu, the temporal evolution of* $\mathbf{s}(t)$ *is modelled by the ODE* $\mathbf{s}'(t) = f(t, \mathbf{s}(t), \mathbf{u}(t))$ *given in equation (1). This ODE is based on a 2D* non-linear *kinematic model where the intruder is assumed to keep constant heading and velocity: the evolution of* $\psi(t)$ *depends only on the evolution of the ownship heading. This corresponds to a degraded mode where the intruder does not perform any collision avoidance maneuver and continues its uniform rectilinear displacement. For simplicity, the velocity of ownship is also considered constant. It is worth noting that* f *is continuous in* t *and* u*, as well as uniformly Lipschitz continuous in* $\mathbf{s}$*. Indeed, its derivative w.r.t.* $\mathbf{s}$ *is bounded on* $\mathbb{R} \times \mathbb{R}^l \times \mathbb{R}^d$ *since both* $v_{own}(t)$ *and* $v_{int}(t)$ *are constants.*

4.3 Neural network based controller

The neural network based controller $\mathcal{N}$ involves a collection of *ReLU neural networks* $\mathbf{N} = \{N^{(1)}, \ldots, N^{(D)}\}$, of which only one is executed at each control step. The network $N_j \in \mathbf{N}$ to be executed at step j is selected based on the command $\mathbf{u}_j$ produced at previous step *i.e.*, $N_j = \lambda(\mathbf{u}_j)$ wherein $\lambda : \mathbf{U} \to \mathbf{N}$ maps every command in $\mathbf{U}$ to a network in $\mathbf{N}$. It is worth noting that all the neural networks in $\mathbf{N}$ are assumed to have been trained already, meaning that they remain unchanged for the run-time of the controller.

Definition 2 *A* ReLU feedforward deep neural network *is a tuple* $N = (L, \{k_l\}_{1 \leq l \leq L}, \mathbf{W}, \mathbf{B})$*. It consists of a* directed acyclic weighted graph *where the nodes are arranged in* L *layers, comprising* $k_1, \ldots, k_L$ *nodes respectively. The first layer is called the* input layer*, the last layer is called the* output layer*, and the layers in between are called the* hidden layers*. Except the input layer, each layer has its nodes connected to the nodes in the preceding layer. More precisely, let* $n_{l,i}$ *be the* i^{th} *node in the* l^{th} *layer. If* $l > 1$*, there exists an edge from* $n_{l-1,j}$ *to* $n_{l,i}$ *for each* $i \in [\![1, k_l]\!]$ *and* $j \in [\![1, k_{l-1}]\!]$*. Moreover, the edge from* $n_{l-1,j}$ *to* $n_{l,i}$ *is assigned a* weight $w^j_{l,i} \in \mathbf{W}$ *and each non-input node* $n_{l,i}$ *is assigned a* bias $b_{l,i} \in \mathbf{B}$*.*

This graph actually corresponds to a function $F : \mathbb{R}^{k_1} \to \mathbb{R}^{k_L}$*. Indeed, each node* $n_{l,i}$ *represents a function* $F_{l,i}$ *of which definition depends on the layer* l*. For the nodes in the input layer, this function is the identity function i.e.,* $F_{1,i} \triangleq \mathrm{id}_{\mathbb{R}}$*,* $\forall i \in [\![1, k_1]\!]$*. For the nodes in the hidden layer* l*, with* $1 < l < L$*, the associated function maps a vector in* $\mathbb{R}^{k_{l-1}}$ *to an element in* $\mathbb{R}$*. It is the composition of a* non-linear ReLU unit $\sigma : x \mapsto \max(0, x)$ *and an* affine transformation *i.e.,* $F_{l,i} : \mathbf{z} \mapsto \sigma\left(\sum_{j=1}^{k_{l-1}} w^j_{l,i} \cdot \mathbf{z}_j + b_{l,i}\right)$*,* $\forall i \in [\![1, k_l]\!]$*. Finally, the function represented by the nodes in the output layer is an affine transformation of a vector in* $\mathbb{R}^{k_{L-1}}$ *i.e.,* $F_{L,i} : \mathbf{z} \mapsto \sum_{j=1}^{k_{L-1}} w^j_{L,i} \cdot \mathbf{z}_j + b_{L,i}$*,* $\forall i \in [\![1, k_L]\!]$*. Overall, the function computed by the* l^{th} *layer of the network is the vector function* $F_l : \mathbf{z} \mapsto (F_{l,1}(\mathbf{z}) \ \ldots \ F_{l,k_l}(\mathbf{z}))^T$ *and the function* F *computed by the network is the composition function* $F \triangleq F_L \circ \ldots \circ F_1$*. In particular,* F *is a deterministic function.*

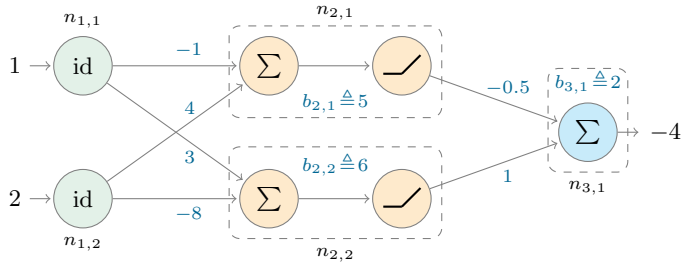


Figure 4: A (tiny) example ReLU network $N = (3, \{2, 2, 1\}, \mathbf{W}, \mathbf{B})$.

In the example of Fig. 4, the input layer yields $F_1((1\ 2)) = (1\ 2)$*, then the hidden layer yields* $F_2((1\ 2)) = (\sigma(-1\times1+4\times2+5)\ \sigma(3\times1-8\times2+6)) = (12\ 0)$*, and the output layer yields* $F_3((12\ 0)) = (-0.5\times12+1\times0+2) = -4$*.*

In addition to the neural networks, the controller $\mathcal{N}$ involves both a pre-processing and a post-processing stage. More precisely, the j^{th} execution of the controller consists of: (i) a *pre-processing* which calculates the input $\mathbf{x}_j \in \mathbb{R}^m$ of the network N_j *i.e.*, $\mathbf{x}_j = \mathrm{Pre}(\mathbf{s}_j)$ wherein $\mathrm{Pre} : \mathbb{R}^l \to \mathbb{R}^m$ (*e.g.*, calculation of a distance from two positions, normalization) (ii) the *neural network execution*, which yields the output vector $\mathbf{y}_j \in \mathbb{R}^p$ such that $\mathbf{y}_j = F_j(\mathbf{x}_j)$ where $F_j : \mathbb{R}^m \to \mathbb{R}^p$ is the function

computed by the network N_j, and (iii) a *post-processing* which determines the command $\mathbf{u}_{j+1}$ given the neural network output $\mathbf{y}_j$ *i.e.*, $\mathbf{u}_{j+1} = \text{Post}(\mathbf{y}_j)$ where $\text{Post} : \mathbb{R}^p \to \mathbf{U}$. Typically, each component $(\mathbf{y}_j)_i \in \mathbb{R}$ of the output $\mathbf{y}_j$ of the network could correspond to a command $\mathbf{u}^{(i)} \in \mathbf{U}$, and the post-processing be $\mathbf{u}_{j+1} = \mathbf{u}^{(k)}$ *s.t.* $k = \underset{i}{\text{argmin}}\left((\mathbf{y}_j)_i\right)$.

Both the pre and post processing are assumed to be deterministic functions, so that the whole controller is also a deterministic function. The overall architecture of $\mathcal{N}$ is illustrated in Fig. 2.

Example 3 *To decide on the maneuver to perform, the ACAS Xu controller uses a collection of* 5 *ReLU networks* $\mathbf{N} = \left\{N^{(1)}, \ldots, N^{(5)}\right\}$. *These networks all have* 6 *hidden layers of* 50 *nodes each. They were each trained with supervised learning to approximate a table of the original ACAS Xu, corresponding to one of the* 5 *possible previous advisories and* $t_{sep} = 0$ *(the* 40 *remaining networks are not considered as they correspond to* $t_{sep} \neq 0$*). Therefore, the function* λ *selecting the network to be executed maps the* 5 *possible advisories to the* 5 *networks in* $\mathbf{N}$. *The pre-processing stage transforms the sampled state* $\mathbf{s}_j$ *into the input* $\mathbf{x}_j$ *of the network by replacing the cartesian coordinates* x, y *into the cylindrical coordinates* ρ, θ *(defined in Fig. 1), and normalizes the resulting vector. The function* $F_j : \mathbb{R}^5 \to \mathbb{R}^5$ *computed by the neural network then outputs* 5 *scores, each one corresponding to a possible maneuver. Finally, the post-processing consists of a argmin function: it chooses the maneuver with the minimal score. A model of the ACAS Xu controller is given in Fig. 5.*

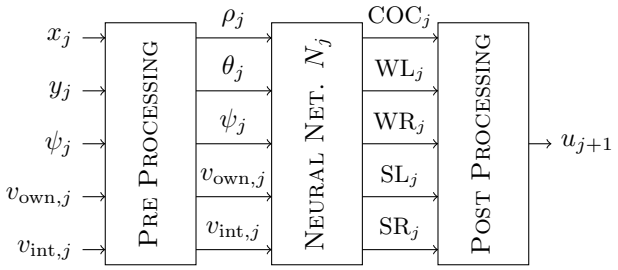


Figure 5: Model of the neural network based ACAS Xu controller.

5 Safety verification problem

In this section, we consider the closed-loop system $\mathcal{C}$ and its evolution over the time horizon τ, the purpose being to prove that no unsafe state can be reached over $[0, \tau]$.

5.1 Reachability definition

Given the deterministic behaviour of the closed-loop system $\mathcal{C}$ (see section 4.1), we define the reachable states of $\mathcal{C}$ as follows:

Definition 3 *The reachable states of the closed-loop system* $\mathcal{C}$ *at a given instant* $t \leq \tau$ *is the set* $\mathbf{R}_t = \{\phi \in \mathbb{R}^l \times \mathbf{U} \cup \{\bot\} \mid \exists \phi_0 \in \mathbf{I},\ \phi = \phi_{\phi_0}(t)\}$ *(see section 4.1 for the definition of* ϕ_{ϕ_0}*).*

Definition 4 *The reachable states of the closed-loop system* $\mathcal{C}$ *for the time interval* $[t_1, t_2] \subset [0, \tau]$ *(resp.* $[t_1, t_2[\subset [0, \tau]$*) is the set* $\mathbf{R}_{[t_1,t_2]} = \{\phi \in \mathbf{R}_t \mid t \in [t_1, t_2]\}$ *(resp.* $\mathbf{R}_{[t_1,t_2[} = \{\phi \in \mathbf{R}_t \mid t \in [t_1, t_2[\}$*).*

5.2 Problem definition

We want to *decide* if, whatever the initial state ϕ_0 in $\mathbf{I}$, the closed-loop system $\mathcal{C}$ remains safe *w.r.t* the set of erroneous states $\mathbf{E}$ over the time horizon τ. In other words, we want to decide if the reachable states of $\mathcal{C}$ in $[0, \tau]$ remain outside $\mathbf{E}$.

Definition 5 *The* safety verification problem $\mathcal{V}$ *consists in deciding if:*

$$\mathbf{R}_{[0,\tau]} \cap \mathbf{E} = \emptyset \tag{2}$$

Reasoning about the problem $\mathcal{V}$ is a difficult task. Indeed, whatever the nature of the controller (based on ReLU networks or not), the problem $\mathcal{V}$ is undecidable when the plant $\mathcal{P}$ has a non-linear dynamics [3, 10] (*e.g.*, ACAS Xu). Furthermore, the neural networks add to the complexity of the verification problem. Indeed, due to the *non-linear* ReLU units and the *many dependencies* induced by the affine transformations, the function computed by a ReLU network is non-monotonic, non convex and highly non-linear. As a result, its behaviour is very difficult to analyze for a *continuum* of inputs, which is the case in problem $\mathcal{V}$ as the initial set $\mathbf{I}$ is infinite. Actually, it has been shown that verifying pre/post-conditions on a ReLU network is a NP-hard problem [12]. Finally, the controller we consider has a non-trivial logic, switching between the networks and involving pre and post-processing stages, which increases the dependencies from one control step to another.

As the problem $\mathcal{V}$ is undecidable, we aim at constructing a *sound* approximation of the reachable states of $\mathcal{C}$. More precisely, we aim at computing a *bounded* set $\widetilde{\mathbf{R}}_{[0,\tau]}$ satisfying $\widetilde{\mathbf{R}}_{[0,\tau]} \supset \mathbf{R}_{[0,\tau]}$. Indeed, provided we are able to compute such a set and if it verifies $\widetilde{\mathbf{R}}_{[0,\tau]} \cap \mathbf{E} = \emptyset$, then (2) is proved to hold. Consequently, we consider the problem $\widetilde{\mathcal{V}}$ defined as follows:

Definition 6 *The* safety verification problem $\widetilde{\mathcal{V}}$ *consists in finding a set* $\widetilde{\mathbf{R}}_{[0,\tau]}$ *satisfying* $\widetilde{\mathbf{R}}_{[0,\tau]} \supset \mathbf{R}_{[0,\tau]}$ *and* $\widetilde{\mathbf{R}}_{[0,\tau]} \cap \mathbf{E} = \emptyset$.

To have a chance to find a solution to problem $\widetilde{\mathcal{V}}$, the set $\widetilde{\mathbf{R}}_{[0,\tau]}$ must be as tight as possible. The next section presents our method for computing a *tight* over-approximation $\widetilde{\mathbf{R}}_{[0,\tau]} \supset \mathbf{R}_{[0,\tau]}$.

6 Reachability-based approach

6.1 Symbolic state and symbolic set

The set $\widetilde{\mathbf{R}}_{[0,\tau]}$ that we aim at constructing is *infinite*. To allow reasoning about this type of set, we introduce the notions of *symbolic state* and *symbolic set*.

Definition 7 *A* symbolic state *is a* 2*-tuple* $([\mathbf{s}], \mathbf{u})$ *wherein* $[\mathbf{s}] \subset \mathbb{R}^l$ *is a* l*-dimensional box i.e., the cartesian product of* l *intervals, and* $\mathbf{u} \in \mathbf{U}$. *It symbolically represents the set* $\{\phi(t) = (\mathbf{s}(t), \mathbf{u}(t)) \in \mathbb{R}^l \times \mathbf{U} \mid \mathbf{s}(t) \in [\mathbf{s}] \ \wedge \ \mathbf{u}(t) = \mathbf{u}\}$.

Example 4 *For the ACAS Xu, the symbolic state* $([\mathbf{s}], u)$ *with* $[\mathbf{s}] = [-20ft, 0ft] \times [8000ft, 8500ft] \times [3.10, 3.14] \times [700ft/s, 700ft/s] \times [600ft/s, 600ft/s]$ *and* $u = 0.0deg/s$ *represents a (infinite) set of states where the intruder is ahead of ownship, moving towards the ownship, and the ACAS Xu controller advises COC.*

Definition 8 *A* symbolic set *is a collection of symbolic states defined by* $\widetilde{\mathbf{\Phi}} = \{([\mathbf{s}]_k, \mathbf{u}_k)\}_{1 \leq k \leq K}$ *wherein* $K \in \mathbb{N}$. *It corresponds to the union of the sets represented by each* $([\mathbf{s}]_k, \mathbf{u}_k)$.

As one can note, a symbolic set can be used to symbolically approximate *any* set of (non-bottom) states of $\mathcal{C}$ (the bottom element is not considered as it does not impact safety). Moreover, our definition yields a rather accurate approximation as it captures the dependency between the state $\mathbf{s}(t)$ of the plant $\mathcal{P}$ and the actuation command $\mathbf{u}(t)$ from the controller. This is made possible as $\mathbf{u}(t)$ can only take a finite number of values.

In the following, we extend the set operations and relations to both symbolic states and symbolic sets *e.g.*, $\phi \in \widetilde{\mathbf{\Phi}}$ iff ϕ belongs to the set represented by $\widetilde{\mathbf{\Phi}}$.

6.2 Over-approximation techniques

Our approach for constructing $\widetilde{\mathbf{R}}_{[0,\tau]}$ is to leverage existing over-approximation techniques. More precisely, we aim at using *validated simulation* to soundly approximate the dynamics of the plant $\mathcal{P}$ and *abstract interpretation* to soundly approximate the behaviour of the controller $\mathcal{N}$. These two techniques are presented below and section 6.3 details how they are combined together to compute $\widetilde{\mathbf{R}}_{[0,\tau]}$.

Validated simulation Let us consider an ODE $\mathbf{s}'(t) = f(t, \mathbf{s}(t), \mathbf{u}(t))$ wherein $\mathbf{s} : \mathbb{R} \to \mathbb{R}^l$, $\mathbf{u} : \mathbb{R} \to \mathbb{R}^d$ is a given function, continous in t, and $f : \mathbb{R} \times \mathbb{R}^l \times \mathbb{R}^d \to \mathbb{R}^l$ is assumed to be continuous in t and $\mathbf{u}$ and uniformly Lipschitz continuous in $\mathbf{s}$. Moreover, let us consider an interval $[t_1, t_2]$ and a l-dimensional box $[\mathbf{s}_{t=t_1}] \subset \mathbb{R}^l$ representing a set of initial values. The goal of validated simulation is to over-approximate the reachable solutions of the ODE satisfying $\mathbf{s}(t = t_1) \in [\mathbf{s}_{t=t_1}]$, over the whole time interval $[t_1, t_2]$. More precisely, it aims at computing the l-box $[\mathbf{s}_{[t_1,t_2]}] \subset \mathbb{R}^l$ approximating the reachable values of $\mathbf{s}(t)$ for $t \in [t_1, t_2]$, and the tighter l-box $[\mathbf{s}_{t=t_2}] \subset [\mathbf{s}_{[t_1,t_2]}]$ approximating the reachable values of $\mathbf{s}(t)$ at $t = t_2$. Consequently, if $\mathbf{s}$ satisfies the ODE and the initial condition $\mathbf{s}(t = t_1) \in [\mathbf{s}_{t=t_1}]$ then $\big(\mathbf{s}(t) \in [\mathbf{s}_{[t_1,t_2]}]\ \forall t \in [t_1, t_2]\big) \wedge (\mathbf{s}(t = t_2) \in [\mathbf{s}_{t=t_2}])$. Usually, validated simulation is based on the 2-step Löhner type algorithm: the enclosure $[\mathbf{s}_{t_1,t_2}]$ is calculated using the Banach fixed point theorem while the enclosure $[\mathbf{s}_{t=t_2}]$ is computed based on a numerical integration method (*e.g.*, Euler, Runge-Kutta) and the associated local truncation error [21].

Abstract interpretation Let us consider a function $F : \mathbb{R}^m \to \mathbb{R}^p$ and let $[\mathbf{x}] \subset \mathbb{R}^m$ be a m-dimensional box representing a set of inputs. The goal of abstract interpretation is to soundly approximate the set of the reachable outputs from $[\mathbf{x}]$ *i.e.*, the set $F([\mathbf{x}]) = \{F(\mathbf{x}) \mid \mathbf{x} \in [\mathbf{x}]\}$. To this end, abstract interpretation leverages an abstract transformer $F^{\#}$ that soundly approximates the semantics of F. Intuitively, it "propagates" $[\mathbf{x}]$ through the function F. This yields the p-box $[\mathbf{y}] = F^{\#}([\mathbf{x}])$ satisfying $[\mathbf{y}] \supset F(\mathbf{X})$. The abstract transformer $F^{\#}$ can rely on interval arithmetics or affine arithmetics for example [15].

6.3 Procedure

In the following, we consider that τ comprises q executions of the controller *i.e.*, $\tau = qT$. The overall idea of our approach is to *iteratively* build the set $\widetilde{\mathbf{R}}_{[0,\tau]}$, based on the successive executions of the controller. To this end, we define a procedure that involves two types of sets:

(a) The symbolic set $\widetilde{\mathbf{R}}_j \supset \mathbf{R}_{jT} \setminus \{\bot\}$ approximates the (non-bottom) reachable states at $t = jT$, with $j \leq q$. The k^{th} symbolic state composing $\widetilde{\mathbf{R}}_j$ is denoted $([\mathbf{s}_j]_k, \mathbf{u}_{j,k})$. It represents a set of states $\mathbf{s}(t)$ that are reachable together with the command $\mathbf{u}_{j,k}$ at $t = jT$.

(b) The symbolic set $\widetilde{\mathbf{R}}_{[j[} \supset \mathbf{R}_{[jT,(j+1)T[} \setminus \{\bot\}$ approximates the (non-bottom) reachable states for $t \in [jT, (j+1)T[$, with $j < q$. The k^{th} symbolic state composing $\widetilde{\mathbf{R}}_{[j[}$ is denoted $([\mathbf{s}_{[j[}]_k, \mathbf{u}_{j,k})$. It represents a set of states $\mathbf{s}(t)$ that are reachable together with the command $\mathbf{u}_{j,k}$ for $t \in [jT, (j+1)T[$.

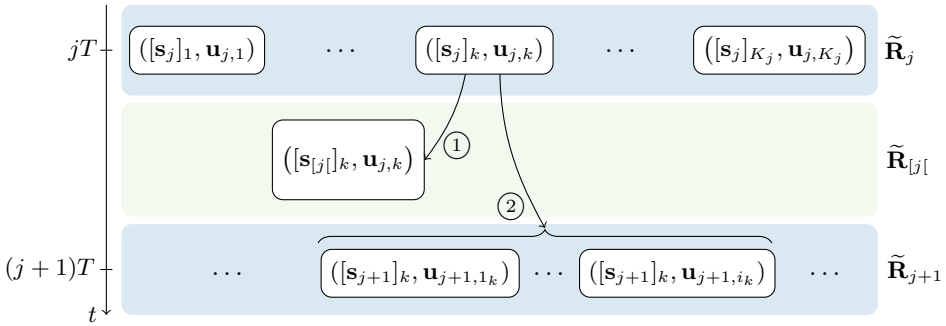


Figure 6: The reachability procedure at control step j, where ① involves validated simulation and ② involves both validated simulation and abstract interpretation.

The procedure starts with the symbolic set $\widetilde{\mathbf{R}}_0 \supset \mathbf{R}_0 = \mathbf{I}$ enclosing the possible initial states. Then, for $j \in [\![0, q-1]\!]$, it computes the reachable symbolic states from each symbolic state $([\mathbf{s}_j]_k, \mathbf{u}_{j,k})$ composing $\widetilde{\mathbf{R}}_j$ (see Fig. 6). More precisely, for each $([\mathbf{s}_j]_k, \mathbf{u}_{j,k}) \in \widetilde{\mathbf{R}}_j$, it computes:

(1) The symbolic state $([\mathbf{s}_{[j[}]_k, \mathbf{u}_{j,k})$ approximating the reachable states from $([\mathbf{s}_j]_k, \mathbf{u}_{j,k})$ over $[jT, (j+1)T[$, where $[\mathbf{s}_{[j[}]_k$ is calculated using validated simulation and $\mathbf{u}_{j,k}$ is the constant

actuation command over $[jT,(j+1)T[$. More specifically, to compute $[\mathbf{s}_{[j[}]_k$, we consider the ODE $\mathbf{s}'(t) = f(t,\mathbf{s}(t),\mathbf{u}(t))$ and the time interval $[jT,(j+1)T]$, with $\mathbf{s}(t=jT) \in [\mathbf{s}_j]_k$ and $\mathbf{u}(t) = \mathbf{u}_{j,k}\ \forall t \in [jT,(j+1)T]$. Validated simulation is used to compute the l-box $[\mathbf{s}_{[jT,(j+1)T]}]$ enclosing the reachable values of $\mathbf{s}(t)$ for $t \in [jT,(j+1)T]$. Then we take $[\mathbf{s}_{[j[}]_k = [\mathbf{s}_{[jT,(j+1)T]}]$ which is sound as $[jT,(j+1)T[\subset [jT,(j+1)T]$.

(2) The symbolic states $([\mathbf{s}_{j+1}]_k, \mathbf{u}_{j+1,1_k}), \ldots, ([\mathbf{s}_{j+1}]_k, \mathbf{u}_{j+1,i_k})$ approximating the reachable states from $([\mathbf{s}_j]_k, \mathbf{u}_{j,k})$ at $t = (j+1)T$, where $[\mathbf{s}_{j+1}]_k$ is calculated using validated simulation and the reachable commands $\mathbf{u}_{j+1,1_k}, \ldots, \mathbf{u}_{j+1,i_k}$ are caluclated using abstract interpretation. More specifically, to compute $[\mathbf{s}_{j+1}]_k$, we consider the same hypotheses as in (1) except that validated simulation is used to compute the l-box $[\mathbf{s}_{t=(j+1)T}]$ enclosing the reachable values of $\mathbf{s}(t)$ at $t = (j+1)T$. Then we take $[\mathbf{s}_{j+1}]_k = [\mathbf{s}_{t=(j+1)T}]$ which is sound even though the actuation command may have changed at $t = (j+1)T$ (this is due to the continuity of $\mathbf{s}$). Additionally, to compute the reachable commands, we approximate the behaviour of the controller as follows. First, the network to be executed $N_{j,k}$ is selected based on the previous command *i.e.*, $N_{j,k} = \lambda(\mathbf{u}_{j,k})$. Then, abstract interpretation is used to compute: (i) the m-box $[\mathbf{x}_j]_k = \mathrm{Pre}^{\#}([\mathbf{s}_j]_k)$ approximating the reachable inputs of the network, (ii) the p-box $[\mathbf{y}_j]_k = F^{\#}_{j,k}([\mathbf{x}_j]_k)$ approximating the reachable outputs of the network, where $F_{j,k}$ denotes the function computed by the network $N_{j,k}$ and (iii) the finite set $\{\mathbf{u}_{j+1,1_k}, \ldots, \mathbf{u}_{j+1,i_k}\} = \mathrm{Post}^{\#}([\mathbf{y}_j])$ approximating the reachable commands at $t = (j+1)T$.

By definition, the stages (1) and (2) yield the symbolic sets $\widetilde{\mathbf{R}}_{[j[}$ and $\widetilde{\mathbf{R}}_{j+1}$, the latter being used in the next iteration. Finally, the q^{th} iteration yields $\widetilde{\mathbf{R}}_{[0,\tau]} = \cup_{0 \leq j < q} \widetilde{\mathbf{R}}_{[j[} \cup \widetilde{\mathbf{R}}_q$ (to be totally rigorous, the bottom element shall be added but this is useless since it does not impact safety).

Actually, to take account of a potential termination of $\mathcal{C}$, we consider a slight variant of the above procedure. Indeed, if a symbolic state $([\mathbf{s}_j]_k, \mathbf{u}_{j,k})$ composing $\widetilde{\mathbf{R}}_j$ satisfies $([\mathbf{s}_j]_k, \mathbf{u}_{j,k}) \subset \mathbf{T}$, then this symbolic state is not further propagated *i.e.*, the reachable symbolic states from $([\mathbf{s}_j]_k, \mathbf{u}_{j,k})$ are not computed. Consequently, if there exists $j^{\mathrm{end}} \leq q$ such that there is no more symbolic state to be propagated from $\widetilde{\mathbf{R}}_{j_{\mathrm{end}}}$, then we take $\widetilde{\mathbf{R}}_{[0,\tau]} = \cup_{0 \leq j < j^{\mathrm{end}}} \widetilde{\mathbf{R}}_{[j[} \cup \widetilde{\mathbf{R}}_{j^{\mathrm{end}}}$. Moreover, if $\widetilde{\mathbf{R}}_{[0,\tau]}$ satisfies $\widetilde{\mathbf{R}}_{[0,\tau]} \cap \mathbf{E} = \emptyset$, then the closed-loop $\mathcal{C}$ is proved to be safe *until it terminates*.

Remark 2 *This mechanism can detect the termination of $\mathcal{C}$ only at the instants $t = T, 2T, 3T, \ldots$, meaning that the true instants when $\mathcal{C}$ terminates are very likely to be missed. However, this remains a good mechanism when $\mathbf{T}$ behaves like an attractor i.e., when $\mathcal{C}$ reaches a state in $\mathbf{T}$ without terminating, then its state tends to stay in $\mathbf{T}$.*

Theorem 1 *The procedure yields a sound approximation of the non-bottom reachable states i.e., $\widetilde{\mathbf{R}}_{[0,\tau]} \supset \mathbf{R}_{[0,\tau]} \setminus \{\bot\}$.*

Proof 2 *The proof is two fold.*

(i) First, let us show by induction that $\widetilde{\mathbf{R}}_j$ soundly approximates the non-bottom reachable states at $t = jT$ i.e., $\widetilde{\mathbf{R}}_j \supset \mathbf{R}_{jT} \setminus \{\bot\}$ for $0 \leq j \leq q$. By definition, $\widetilde{\mathbf{R}}_0 \supset \mathbf{R}_0 = \mathbf{I}$ is a sound approximation of the non-bottom reachable states at $t = 0$. Let $\phi^* \neq \bot$ be a reachable state at $t^* = (j+1)T$. Given the definition of a reachable state (see definition 3), there exists a unique function $\phi^*_{\phi_0} : [0,\tau] \to \mathbb{R}^l \times \mathbf{U} \cup \{\bot\}$ such that $\phi^* = \phi^*_{\phi_0}((j+1)T)$. Moreover, since $\phi^* \neq \bot$, neither the target set $\mathbf{T}$ nor the bottom state have been reached already i.e., $\phi^*_{\phi_0}(t) \notin \mathbf{T} \cup \{\bot\}\ \forall t < (j+1)T$. By induction, there is a symbolic state $([\mathbf{s}_j]_k, \mathbf{u}_{j,k}) \in \widetilde{\mathbf{R}}_j$ such that $\phi^*_{\phi_0}(jT) \in ([\mathbf{s}_j]_k, \mathbf{u}_{j,k})$. Additionally, as $\phi^*_{\phi_0}(jT) \notin \mathbf{T}$, $([\mathbf{s}_j]_k, \mathbf{u}_{j,k})$ satisfies $([\mathbf{s}_j]_k, \mathbf{u}_{j,k}) \not\subset \mathbf{T}$. Consequently, the procedure computes the reachable states from $([\mathbf{s}_j]_k, \mathbf{u}_{j,k})$ at $t = (j+1)T$, yielding the symbolic states $([\mathbf{s}_{j+1}]_k, \mathbf{u}_{j+1,1_k}), \ldots, ([\mathbf{s}_{j+1}]_k, \mathbf{u}_{j+1,i_k})$. As validated simulation and abstract interpretation are sound, these symbolic states constitute a sound approximation. Therefore, $\phi^*_{\phi_0}((j+1)T) \in ([\mathbf{s}_{j+1}]_k, \mathbf{u}_{j+1,1_k}), \ldots, ([\mathbf{s}_{j+1}]_k, \mathbf{u}_{j+1,i_k}) \subset \widetilde{\mathbf{R}}_{j+1}$. Hence $\phi^* \in \widetilde{\mathbf{R}}_{j+1}$.

(ii) Secondly, let us show that $\widetilde{\mathbf{R}}_{[j[}$ soundly approximates the non-bottom reachable states over the time interval $[jT,(j+1)T[$ i.e., $\widetilde{\mathbf{R}}_{[j[} \supset \mathbf{R}_{[jT,(j+1)T[} \setminus \{\bot\}$ for $0 \leq j < q$. Let $\phi^* \neq \bot$ be a reachable state at $t^* \in [jT,(j+1)T[$. There exists a unique function $\phi^*_{\phi_0}$ such that

$\phi^* = \phi^*_{\phi_0}(t^*)$ and $\phi^*_{\phi_0}(t) \notin \mathbf{T} \cup \{\bot\}$ $\forall t < t^*$. As shown before, $\phi^*_{\phi_0}(jT) \in \widetilde{\mathbf{R}}_j$, so there is a symbolic state $([\mathbf{s}_j]_k, \mathbf{u}_{j,k}) \in \widetilde{\mathbf{R}}_j$ such that $\phi^*_{\phi_0}(jT) \in ([\mathbf{s}_j]_k, \mathbf{u}_{j,k})$. Additionally, since $\phi^*_{\phi_0}(jT) \notin \mathbf{T}$, $([\mathbf{s}_j]_k, \mathbf{u}_{j,k})$ satisfies $([\mathbf{s}_j]_k, \mathbf{u}_{j,k}) \not\subset \mathbf{T}$. Consequently, the procedure computes the reachable states from $([\mathbf{s}_j]_k, \mathbf{u}_{j,k})$ over the interval $[jT, (j+1)T[$, yielding the symbolic state $([\mathbf{s}_{[j[}]_k, \mathbf{u}_{j,k})$. As validated simulation is sound and the command signal remains constant over the interval $[jT, (j+1)T[$, this symbolic state constitutes a sound approximation. Therefore, $\phi^*_{\phi_0}(t^*) \in ([\mathbf{s}_{[j[}]_k, \mathbf{u}_{j,k}) \subset \widetilde{\mathbf{R}}_{[j[}$. Hence $\phi^* \in \widetilde{\mathbf{R}}_{[j[}$.

□

6.4 Optimizations

Improving precision In the above procedure, a *single* l-box $[\mathbf{s}_{[j[}]_k$ encloses the reachable states $\mathbf{s}(t)$ from the symbolic state $([\mathbf{s}_j]_k, \mathbf{u}_{j,k})$ over $[jT, (j+1)T[$. Due to its shape, the l-box $[\mathbf{s}_{[j[}]_k$ may contain a lot of unreachable states, resulting in a loose approximation (see Fig. 7). In order to yield a tighter approximation, the procedure is slightly modified. Instead of using a single l-box to approximate the reachable states $\mathbf{s}(t)$, a collection of $M > 1$ l-boxes is used. This collection of l-boxes is obtained by performing M integration steps *i.e.*, M successive validated simulations. More precisely, we start with the l-box $[\mathbf{s}_{j,0}]_k \triangleq [\mathbf{s}_j]_k$. Then, for $i \in [\![0, M-1]\!]$, we consider the ODE $\mathbf{s}'(t) = f(t, \mathbf{s}(t), \mathbf{u}(t))$ and the time interval $\left[\left(j + \frac{i}{M}\right) T, \left(j + \frac{i+1}{M}\right) T\right]$, with $\mathbf{s}\left(t = \left(j + \frac{i}{M}\right) T\right) \in [\mathbf{s}_{j,i}]_k$ and $\mathbf{u}(t) = \mathbf{u}_{j,k}$ $\forall t \in \left[\left(j + \frac{i}{M}\right) T, \left(j + \frac{i+1}{M}\right) T\right]$. Validated simulation is used to compute (1) the l-box $[\mathbf{s}_{[j[,i]}]_k$ approximating the reachable states $\mathbf{s}(t)$ over $\left[\left(j + \frac{i}{M}\right) T, \left(j + \frac{i+1}{M}\right) T\right[$ and (2) the l-box $[\mathbf{s}_{j,i+1}]_k$ enclosing the reachable states $\mathbf{s}(t)$ at $t = \left(j + \frac{i+1}{M}\right) T$. The latter l-box is used to perform next integration step. Finally, we take $[\mathbf{s}_{[j[}]_k = \left\{[\mathbf{s}_{[j[,i]}]_k\right\}_{0 \leq i < M}$ and also $[\mathbf{s}_{j+1}]_k = [\mathbf{s}_{j,M}]_k$.

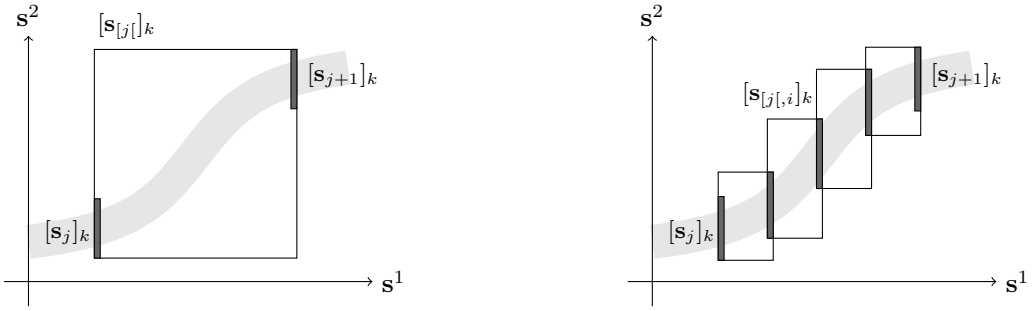


Figure 7: Over-approximation of the plant dynamics using validated simulation with a single integration step (left) and $M = 4$ integration steps (right).

Algorithm 1 Over-approximation of the plant dynamics.

Input: The function f describing the plant dynamics, the execution period T of the controller, the number of integration steps M and the symbolic state $([\mathbf{s}_j]_k, \mathbf{u}_{j,k})$.
Output: The l-boxes $[\mathbf{s}_{[j[}]_k$ and $[\mathbf{s}_{j+1}]_k$.

```
function SIMULATE(f, T, M, ([s_j]_k, u_{j,k}))
    [s_{j,0}]_k ← [s_j]_k
    for i ∈ [[0, M − 1]] do
        ([s_{[j[,i]}]_k, [s_{j,i+1}]_k) ← validatedSimulation(f, [(j + i/M) T, (j + (i+1)/M) T], [s_{j,i}]_k, u_{j,k})
    end for
    [s_{[j[}]_k ← {[s_{[j[,0]}]_k, ..., [s_{[j[,M−1]}]_k}
    [s_{j+1}]_k ← [s_{j,M}]_k
    return ([s_{[j[}]_k, [s_{j+1}]_k)
end function
```

Improving time complexity In the worst case, the number of symbolic states in $\widetilde{\mathbf{R}}_j$ grows exponentially with j. Indeed, each symbolic state $([\mathbf{s}_j]_k, \mathbf{u}_{j,k})$ composing $\widetilde{\mathbf{R}}_j$ can lead up to P symbolic states $([\mathbf{s}_{j+1}]_k, \mathbf{u}_{j+1,1_k}), \ldots, ([\mathbf{s}_{j+1}]_k, \mathbf{u}_{j+1,i_k})$ in $\widetilde{\mathbf{R}}_{j+1}$ (recall that P is the number of

elements in $\mathbf{U}$ *i.e.*, the number of possible actuation commands). In order to avoid an exponential blow up, the procedure is slightly modified by keeping the number of symbolic states in $\widetilde{\mathbf{R}}_j$ below a given threshold Γ, for all $j \in [\![0, q]\!]$. As a result, provided we can bound the execution time of validated simulation and abstract interpretation, the time complexity of the procedure remains linear with q. For keeping the size of $\widetilde{\mathbf{R}}_j$ below Γ, some symbolic states are *joined* based on a dedicated heuristics. This heuristics uses the notion of *distance* between two symbolic states as well as a *join* operation.

Definition 9 *The* distance *between two symbolic states* $([\mathbf{s}]_1, \mathbf{u})$ *and* $([\mathbf{s}]_2, \mathbf{u})$ *with* same *actuation command* $\mathbf{u}$ *is defined as the euclidean distance between the centers of the* l*-boxes* $[\mathbf{s}]_1$ *and* $[\mathbf{s}]_2$*:*

$$d\left(([\mathbf{s}]_1, \mathbf{u}), ([\mathbf{s}]_2, \mathbf{u})\right) = ||C_1 - C_2||_2^2 \tag{3}$$

wherein C_1 *(resp* C_2*) is a* l*-dimensional vector of which the* i^{th} *component is the center of the* i^{th} *interval composing* $[\mathbf{s}]_1$ *(resp* $[\mathbf{s}]_2$*).*

Definition 10 *The* join *operation takes as inputs two symbolic states* $([\mathbf{s}]_1, \mathbf{u})$ *and* $([\mathbf{s}]_2, \mathbf{u})$ *with* same *actuation command* $\mathbf{u}$ *and outputs a symbolic state* $([\mathbf{s}]_3, \mathbf{u})$ *such that* $[\mathbf{s}]_3$ *is the smaller* l*-box containing both* $[\mathbf{s}]_1$ *and* $[\mathbf{s}]_2$*.*

The heuristics works as follows. At the j^{th} control step, if the number of symbolic states K_j in $\widetilde{\mathbf{R}}_j$ is greater than Γ, then the symbolic states composing $\widetilde{\mathbf{R}}_j$ are clustered into P groups, each one corresponding to a given actuation command. More specifically, the i^{th} group is $\mathscr{G}_i = \left\{([\mathbf{s}_j]_k, \mathbf{u}_{j,k}) \in \widetilde{\mathbf{R}}_j \mid \mathbf{u}_{j,k} = \mathbf{u}^{(i)}\right\}$ where $\mathbf{u}^{(i)}$ is the i^{th} element of the set $\mathbf{U}$ of the possible actuation commands (see section 4.1). For each group $\mathscr{G}_i$, a distance matrix $\mathscr{D}_i$ is calculated based on definition 9. Then, the distance matrices $\mathscr{D}_1, \ldots, \mathscr{D}_P$ are used to identify the two *closest* symbolic states in $\widetilde{\mathbf{R}}_j$ (note that these two closest symbolic states necessarily have the same actuation command). Finally, using the join operation introduced in definition 10, the two closest symbolic states are joined. The set $\widetilde{\mathbf{R}}_j$ is updated accordingly and the process is repeated until $K_j \leq \Gamma$ (see Algorithm 2).

The choice of the threshold Γ allows a trade-off between accuracy (large Γ) and computational efficiency (small Γ).

Algorithm 2 Heuristics for keeping the number of symbolic states in $\widetilde{\mathbf{R}}_j$ below the threshold Γ.

Input: The symbolic set $\widetilde{\mathbf{R}}_j$ and the threshold Γ.
Ensure: length$(\widetilde{\mathbf{R}}_j) \leq \Gamma$ and $\widetilde{\mathbf{R}}_j \supset$ old$(\widetilde{\mathbf{R}}_j)$.

procedure RESIZE($\widetilde{\mathbf{R}}_j$, Γ)
  $K_j \leftarrow$ length$(\widetilde{\mathbf{R}}_j)$
  while $K_j > \Gamma$ **do**
    for $i \in [\![1, P]\!]$ **do**
      $\mathscr{G}_i \leftarrow \left\{([\mathbf{s}_j]_k, \mathbf{u}_{j,k}) \in \widetilde{\mathbf{R}}_j \mid \mathbf{u}_{j,k} = \mathbf{u}^{(i)}\right\}$
      $\mathscr{D}_i \leftarrow$ calculateDistanceMatrix$(\mathscr{G}_i)$
    end for
    $(([\mathbf{s}_j]_{k_1}, \mathbf{u}_{j,k_1}), ([\mathbf{s}_j]_{k_2}, \mathbf{u}_{j,k_2})) \leftarrow$ findClosestStates$(\mathscr{D}_1, \ldots, \mathscr{D}_P)$ ▷ $\mathbf{u}_{j,k_1} = \mathbf{u}_{j,k_2}$
    $([\mathbf{s}_j]_{k_3}, \mathbf{u}_{j,k_3}) \leftarrow$ join$(([\mathbf{s}_j]_{k_1}, \mathbf{u}_{j,k_1}), ([\mathbf{s}_j]_{k_2}, \mathbf{u}_{j,k_2}))$ ▷ $\mathbf{u}_{j,k_3} = \mathbf{u}_{j,k_1}$
    $\widetilde{\mathbf{R}}_j \leftarrow \left(\widetilde{\mathbf{R}}_j \setminus \{([\mathbf{s}_j]_{k_1}, \mathbf{u}_{j,k_1}), ([\mathbf{s}_j]_{k_2}, \mathbf{u}_{j,k_2})\}\right) \cup \{([\mathbf{s}_j]_{k_3}, \mathbf{u}_{j,k_3})\}$
    $K_j \leftarrow K_j - 1$
  end while
end procedure

Remark 3 *As one may note,* Γ *must be chosen greater than* P*. Indeed, two symbolic states with two different actuation commands cannot be joined so the heuristics would fail to keep the number of symbolic states in* $\widetilde{\mathbf{R}}_j$ *strictly below* P*.*

6.5 Overall algorithm

Algorithm 3 Reachability analysis of the closed-loop system $\mathcal{C}$.

Input: The closed-loop system $\mathcal{C} = (\mathcal{P}, \mathcal{N})$, the approximated set of the initial states $\widetilde{\mathbf{R}}_0 \supseteq \mathbf{R}_0 = \mathbf{I}$, the set of the erroneous states $\mathbf{E}$, the target set $\mathbf{T}$, the number of control steps q, the number of integration steps M and the threshold Γ.

Output: A Boolean indicating whether the closed-loop $\mathcal{C}$ is proved safe until it terminates.

$j_{\text{end}} \leftarrow q$
hasTerminated $\leftarrow$ False
for $j \in [\![0, q-1]\!]$ **do**
  ▷ keep the size of $\widetilde{\mathbf{R}}_j$ below Γ (see Algorithm 2)
  RESIZE($\widetilde{\mathbf{R}}_j$,Γ)
  ▷ compute the symbolic sets $\widetilde{\mathbf{R}}_{[j[}$ and $\widetilde{\mathbf{R}}_{j+1}$
  $\widetilde{\mathbf{R}}_{[j[} \leftarrow \emptyset$
  $\widetilde{\mathbf{R}}_{j+1} \leftarrow \emptyset$
  for $\Big(([\mathbf{s}_j]_k, \mathbf{u}_{j,k}) \in \widetilde{\mathbf{R}}_j \ \wedge \ ([\mathbf{s}_j]_k, \mathbf{u}_{j,k}) \not\subset \mathbf{T}\Big)$ **do**
    ▷ approximate the dynamics of the plant (see Algorithm 1)
    $\left([\mathbf{s}_{[j[}]_k, [\mathbf{s}_{j+1}]_k\right) \leftarrow$ SIMULATE$(f, T, M, ([\mathbf{s}_j]_k, \mathbf{u}_{j,k}))$
    ▷ approximate the behaviour of the controller
    $N_{j,k} \leftarrow \lambda\left(\mathbf{u}_{j,k}\right)$
    $[\mathbf{x}_j]_k \leftarrow \text{Pre}^{\#}\left([\mathbf{s}_j]_k\right)$
    $[\mathbf{y}_j]_k \leftarrow F^{\#}_{j,k}\left([\mathbf{x}_j]_k\right)$
    $\{\mathbf{u}_{j+1,1_k}, \ldots, \mathbf{u}_{j+1,i_k}\} \leftarrow \text{Post}^{\#}\left([\mathbf{y}_j]_k\right)$
    ▷ update the symbolic set $\widetilde{\mathbf{R}}_{[j[}$ approximating the reachable states over $[jT, (j+1)T[$
    $\widetilde{\mathbf{R}}_{[j[} \leftarrow \widetilde{\mathbf{R}}_{[j[} \cup \{([\mathbf{s}_{[j[}]_k, \mathbf{u}_{j,k})\}$
    ▷ update the symbolic set $\widetilde{\mathbf{R}}_{j+1}$ approximating the reachable states at $t = (j+1)T$
    $\widetilde{\mathbf{R}}_{j+1} \leftarrow \widetilde{\mathbf{R}}_{j+1} \cup \{([\mathbf{s}_{j+1}]_k, \mathbf{u}_{j+1,1_k}), \ldots, ([\mathbf{s}_{j+1}]_k, \mathbf{u}_{j+1,i_k})\}$
  end for
  ▷ check for termination
  if $\widetilde{\mathbf{R}}_{j+1} \subset \mathbf{T}$ **then**
    $j_{\text{end}} \leftarrow j + 1$
    hasTerminated $\leftarrow$ True
    break
  end if
end for
▷ construct the symbolic set $\widetilde{\mathbf{R}}_{[0,\tau]}$
$\widetilde{\mathbf{R}}_{[0,\tau]} \leftarrow \widetilde{\mathbf{R}}_{[0[} \cup \ldots \cup \widetilde{\mathbf{R}}_{[j_{\text{end}}-1[} \cup \widetilde{\mathbf{R}}_{j_{\text{end}}}$
return $\Big(\widetilde{\mathbf{R}}_{[0,\tau]} \cap \mathbf{E} = \emptyset \ \wedge \ \text{hasTerminated}\Big)$

6.6 Implementation details

We implemented our procedure as a Python program that interfaces with existing tools. The validated simulation of the plant dynamics is based on DynIBEX [21]. The abstract transformers $\text{Pre}^{\#}$ and $\text{Post}^{\#}$ approximating the semantics of the pre- and post-processing functions are based on interval arithmetics, which has the advantage to be easy to implement and to offer a computationally efficient analysis while still being accurate for simple functions. The abstract transformer of the neural network function $F^{\#}_{j,k}$ relies on a dedicated tool named ReluVal [25], which uses interval arithmetics together with symbolic interval propagation.

7 Experiments

7.1 Experimental setup

Partitioning For verifying the ACAS Xu, we used an empirical partitioning of the possible initial states. More precisely, the circle $\mathcal{R}$ representing the possible initial positions (x_0, y_0) of the intruder was partitioned into 629 arcs of length 80 ft each. Additionally, for each arc, the possible initial headings ψ_0 of the intruder were partitioned into 316 subsets of size 0.01 rad each (see Fig 8). With the initial velocities $v_{\text{own},0}$ and $v_{\text{int},0}$ being fixed, we obtained a partition of size $K_0 = 198,764$ of the possible initial states $\mathbf{s}_0$ of the plant $\mathcal{P}$. Then, each element of this partition was over-approximated by a 5-dimensional box $[\mathbf{s}_0]_k \subset \mathbb{R}^5$, with $1 \leq k \leq K_0$. Finally, we took as input for the procedure the symbolic set $\widetilde{\mathbf{R}}_0 = \{([\mathbf{s}_0]_k, 0.0 \text{ deg/s})\}_{1 \leq k \leq K_0}$.

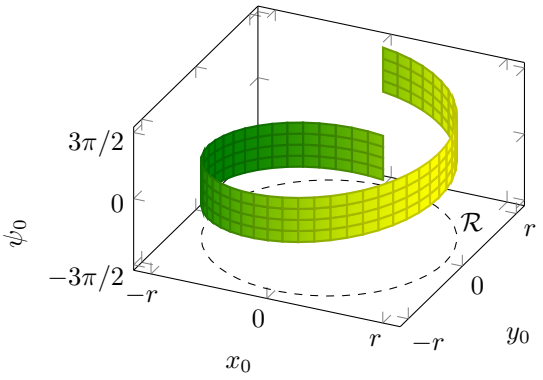


Figure 8: The ribbon-like set of the possible initial states (x_0, y_0, ψ_0) with an example of partition represented as a mesh grid.

The reason for partitioning was three fold. First, a single initial symbolic state $([\mathbf{s}_0], 0.0 \text{ deg/s})$ approximating $\mathbf{I}$ necessarily contains the unsafe set $\mathbf{E}$, due to the shape of $[\mathbf{s}_0]$ (a box containing the circle $\mathcal{R}$ also contains the center of $\mathcal{R}$ corresponding to a collision between the two aircraft). Secondly, the K_0 initial symbolic states composing $\widetilde{\mathbf{R}}_0$ can be seen as K_0 *independent* verification problems, of which resolution can be parallelized. Finally, the smaller the box $[\mathbf{s}_0]_k$, the more precise the reachability analysis since the function f representing the dynamics of $\mathcal{P}$ is uniformly Lipschitz continuous in $\mathbf{s}$ and the functions computed by the neural networks are also uniformly Lipschitz continuous [25].

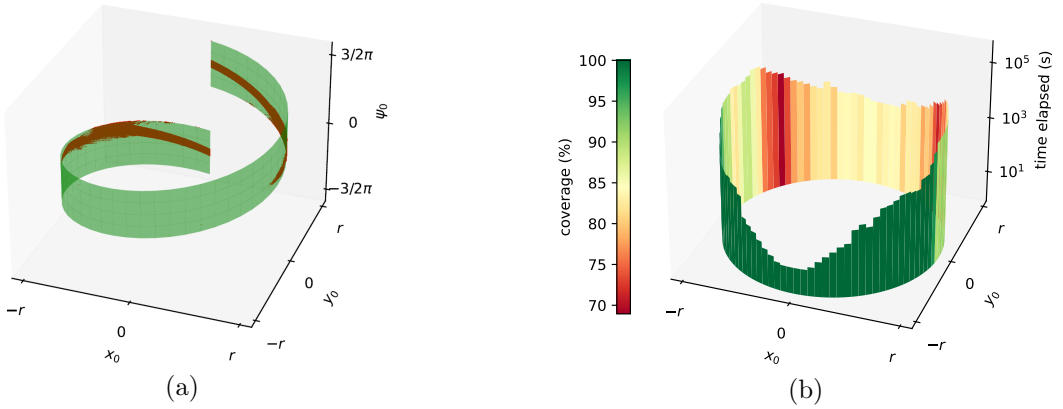
Split refinement For the same reason as mentioned above, when the system could not be proved safe for a given initial symbolic state $([\mathbf{s}_0]_k, 0.0 \text{ deg/s})$, then this initial symbolic state was splitted into smaller initial symbolic states, leading to a new reachability analysis. More precisely, $[\mathbf{s}_0]_k$ was bisected along the dimensions corresponding to x_0, y_0 and ψ_0, yielding 2^3 new initial symbolic states. This split refinement process was repeated iteratively until the system could be proved safe, with a maximum depth of 2.

The experiment was conducted using $M = 10$ for the number of integration steps and $\Gamma = P = 5$ for the threshold on the number of symbolic states in $\widetilde{\mathbf{R}}_j$. Moreover, it was run on CentOS 7 with 2 Intel® Xeon® processors E5-2670 v3 @ 2.30GHz of 12 cores (24 threads) each and 64 GB RAM.

7.2 Results

In our experiment, we recorded (1) the *time elapsed i.e.*, the time necessary for performing the reachability analysis and (2) the *coverage* c representing the percentage of the possible initial states for which the ACAS Xu was proved safe until it terminates. More precisely, the coverage c was calculated as follows: $c = 100/K_0 \cdot \sum_{d=0}^{2} n_d/(2^3)^d$ wherein n_d is the number of initial symbolic states resulting from d split refinements and for which the ACAS Xu was proved safe. The reachability analysis took about 12 days and yielded a coverage $c = 90.3\%$, meaning that the ACAS Xu was proved safe for 90.3% of the possible initial states.

Although we did not obtain a complete proof of safety, we could leverage the partition of the set $\mathbf{I}$ to identify the initial states for which the ACAS Xu was proved safe and the initial states



(a) (b)

Figure 9: (a) The initial states for which the system was proved safe (in green) and the initial states for which the system could not be proved safe (in red), (b) The coverage and time elapsed *w.r.t.* the initial position of the intruder (each bar corresponds to a subset of the initial states where the position (x_0, y_0) of the intruder lies along an arc of length 500ft).

for which it could not be proved safe (see Fig 9.a). It is worth noting that such a result represents a valuable information from a practical point of view. For instance, it could be used to design a real-time monitoring mechanism that switches to a more robust controller if the system encounters an initial state for which it was not proved safe.

The results that we obtained by partitioning the set $\mathbf{I}$ also constitute a valuable information in terms of "*explainability*", in the sense that they help understanding the behaviour of the overall system. For instance, as one can see in Fig 9.b, the initial states that yielded the hardest verification tasks correspond to the cases where the intruder is approaching from the left ($x_0 < 0 \ \wedge \ y_0 > 0$) or approaching from the right ($x_0 > 0 \ \wedge \ y_0 > 0$). Indeed, the coverage obtained in these regions is around 75% while it ranges from 85% to 100% elsewhere. Additionally, in these regions, the time necessary for performing the reachability analysis is about $5 \cdot 10^4 s$ while it is around or below $10^3 s$ elsewhere. Such a result provides an interesting information about the potential weaknesses of the controller, which can be interpreted at the system level. It suggests that the most critical situations are encountered not when the intruder is directly ahead of the ownship but when it approaches from the left or from the right. In addition to representing a valuable knowledge about the behaviour of the system, this information could be used to generate new data with the aim of retraining the networks for example. Furthermore, as one can see in Fig. 9.b, the results are roughly symmetrical *w.r.t.* the $x_0 = 0$ axis, both in terms of coverage and time elapsed, which suggests that the system has a similar behaviour for two initial states that are symmetrical *w.r.t.* the $x_0 = 0$ axis. It is worth noting that such a behaviour is quite consistent since the collision avoidance problem is totally symmetrical *w.r.t.* the $x_0 = 0$ axis. We believe that such an information can help building "*trust*" in the overall system.

8 Conclusion and future work

This paper presented a technique to verify the safety requirements of complex neural network controlled systems such as the ACAS Xu. The proposed technique leverages a generic model of a neural network controlled system together with a reachability analysis, combining validated simulation and abstract interpretation. We evaluated the applicability of our approach by providing the first sound guarantees of safety of the overall neural network based ACAS Xu. Although we could not obtain a complete proof of safety, we showed that our approach can provide valuable information from a practical point of view.

For future work, instead of using a uniform, empirically-generated partition of the initial states, we aim at finding a more efficient partitioning strategy. For example, we could explore the techniques employed in similar problems such as meshing generation in computational fluid dynamics. Another direction is to propose an efficient heuristics for splitting the initial symbolic states when the system cannot be proved safe. Instead of using a simple bisection along each dimension, we could identify the variable having the most influence on the overall system behaviour, and split

along the corresponding dimension only. A third direction is to combine our approach with an efficient falsification strategy that can search for unsafe trajectories when the system cannot be proved safe. Lastly, for the ACAS Xu, we could consider multiple UAVs, each one being equipped with a collision avoidance controller. Indeed, our model and procedure only have to be slightly adapted to represent multiple agents interacting together, all equipped with a controller. The plant could capture the dynamics of the multiple agents (the same way we captured the dynamics of both the ownship and the intruder) and be combined with several controllers. Then, instead of evaluating one controller, our procedure would evaluate several controllers, which is straightforward if all the controllers execute in the same time interval.

References

[1] EUROCAE ED-79A/SAE ARP 4754A. Guidelines for development of civil aircraft and systems. 2010.

[2] Matthias Althoff. An introduction to CORA 2015. In *Proceedings of the 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems*, ARCH'15, pages 120–151, 2015.

[3] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, A. Olivero X. Nicollin, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. In *Theor. Comput. Sci. 138, 1*, page 3–34, 1995.

[4] Sergiy Bogomolov, Goran Frehse, Amit Gurung, Dongxu Li, Georg Martius, and Rajarshi Ray. Falsification of hybrid systems using symbolic reachability and trajectory splicing. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, page 1–10, 2019.

[5] Chen C., Seff A., Kornhauser A., and Xiao J. DeepDriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the 2015 IEEE International Conference on Computer Vision*, ICCV, pages 2722–2730, 2015.

[6] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Proceedings of the 25th International Conference on Computer Aided Verification*, CAV 2013, page 258–263, 2013.

[7] Julian Kyle D. and Kochenderfer Mykel J. Guaranteeing safety for neural network-based aircraft collision avoidance systems. In *Proceedings of the 2019 IEEE/AIAA 38th Digital Avionics Systems Conference*, DASC, 2019.

[8] EUROCAE ED-12C/RTCA DO-178C. Software considerations in airborne systems and equipment certification. 2012.

[9] S. Dutta, X. Chen, and S. 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 '19, page 157–168, 2019.

[10] Hainry E. Reachability in linear dynamical systems. In *Proceedings of the 4th Conference on Computability in Europe*, CiE 2008, pages 241–250, 2008.

[11] Tran HD. et al. NNV: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *Proceedings of the 32nd International Conference on Computer Aided Verification*, CAV 2020, pages 3–17, 2020.

[12] Katz G., Barrett C., Dill D.L., Julian K., and Kochenderfer M.J. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Proceedings of the 29th International Conference on Computer Aided Verification*, CAV 2017, pages 97–117, 2017.

[13] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev. AI2: Safety and robustness certification of neural networks with abstract interpretation. In *Proceedings of the 2018 IEEE Symposium on Security and Privacy*, SP, pages 3–18, 2018.

[14] Chao Huang, Jiameng Fan, Wenchao Li, Xin Chen, and Qi Zhu. ReachNN: Reachability analysis of neural-network controlled systems. In *ACM Transactions on Embedded Computing Systems 18, 5s, Article 106*, 2019.

[15] Stolfi J. and Figueiredo L. An introduction to affine arithmetic. In *Trends in Applied and Computational Mathematics, 4, 297-312*, 2003.

[16] Julian K.D., Kochenderfer M.J., and Owen M.P. Deep neural network compression for aircraft collision avoidance systems. *ArXiv, abs/1810.04240*, 2018.

[17] Bojarski M., Testa D., Dworakowski D., Firner B., Flepp B., Goyal P., Jackel L., Monfort M., Muller U., Zhang J., Zhang X., and Zhao J.and Zieba K. End to end learning for self-driving cars. *ArXiv, abs/1604.07316*, 2016.

[18] G. Manfredi and Y. Jestin. An introduction to ACAS Xu and the challenges ahead. In *Proceedings of the IEEE/AIAA 35th Digital Avionics Systems Conference*, DASC, pages 1–9, 2016.

[19] Ehlers R. Formal verification of piece-wise linear feed-forward neural networks. In *Proceedings of the 15th International Symposium on Automated Technology for Verification and Analysis*, ATVA 2017, pages 269–286, 2017.

[20] Ivanov R., Weimer J., Alur R., Pappas G.J., and Lee I. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC'19, 2019.

[21] Julien Alexandre Dit Sandretto and Alexandre Chapoutot. Validated explicit and implicit runge-kutta methods. In *Reliable Computing electronic edition, 2016, Special issue devoted to material presented at SWIM 2015, 22*, 2015.

[22] EUROCAE WG 75.1 / RTCA SC-147. Minimum operational performance standards for airborne collision avoidance. 2020.

[23] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 10825–10836, 2018.

[24] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. In *Proceedings of the ACM on Programming Languages 3, POPL, Article 41*, 2019.

[25] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. In *Formal security analysis of neural networks using symbolic intervals*, Proceedings of the 27th USENIX Conference on Security Symposium, page 1599–1614, SEC'18.

[26] Huang X., Kroening D., Ruan W., Ruan W., Sun Y., Thamo E., Wu M., and Yi X. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. In *Comput. Sci. Rev., 37, 100270*, 2020.

[27] Annpureddy Y., Liu C., Fainekos G., and Sankaranarayanan S. S-TaLiRo: A tool for temporal logic falsification for hybrid systems. In *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS 2011, pages 254–257, 2011.

[28] Pan Y., Cheng C., Saigol K., Lee K., Yan X., Theodorou E., and Boots B. Agile autonomous driving using end-to-end deep imitation learning. In *Robotics: Science and Systems*, 2018.