Formal Verification of Neural Agents in Non-deterministic Environments

Michael E. Akintunde, Elena Botoeva, Panagiotis Kouvaros, Alessio Lomuscio
Imperial College London
London, United Kingdom
{michael.akintunde13,e.botoeva,p.kouvaros,a.lomuscio}@imperial.ac.uk

ABSTRACT

We introduce a model for agent-environment systems where the agents are implemented via feed-forward ReLU neural networks and the environment is non-deterministic. We study the verification problem of such systems against CTL properties. We show that verifying these systems against reachability properties is undecidable. We introduce a bounded fragment of CTL, show its usefulness in identifying shallow bugs in the system, and prove that the verification problem against specifications in bounded CTL is in coNExpTime and PSpace-hard. We present a novel parallel algorithm for MILP-based verification of agent-environment systems, present an implementation, and report the experimental results obtained against a variant of the VerticalCAS use-case.

KEYWORDS

Verification; Neural Systems

ACM Reference Format:
Michael E. Akintunde, Elena Botoeva, Panagiotis Kouvaros, Alessio Lomuscio. 2020. Formal Verification of Neural Agents in Non-deterministic Environments. In *Proc.of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020,* IFAAMAS, 9 pages.

1 INTRODUCTION

Forthcoming autonomous and robotic systems, including autonomous vehicles, are expected to use machine learning (ML) methods for some of their components. Differently from more conventional AI systems that are programmed directly by engineers, components based on ML are synthesised from data and implemented via neural networks. In an autonomous system these components could execute functions such as perception [24, 33] and control [18, 20]. Employing ML components has considerable attractions in terms of performance (e.g., image classifiers), and, sometimes, ease of realisation (e.g., non-linear controllers). However, it also raises concerns in terms of overall system safety. Indeed, it is known that neural networks, as presently used, are fragile and hard to understand [35].

If ML components are to be used in safety-critical systems, including various forthcoming autonomous systems, it is essential that they are verified and validated before deployment; standard practice for conventional software. In some areas of AI, notably multi-agent systems (MAS), considerable research has already addressed the automatic verification of AI systems [13, 26]. These concern the validation of either MAS models [23, 31], or MAS programs [4, 9] against expressive AI-inspired specifications, such as those expressible in epistemic and strategy logic. However, with the exceptions discussed below, there is little work addressing the verification of AI systems synthesised from data and implemented via neural networks. This paper makes a contribution in this direction.

Specifically, we formalise and analyse a closed-loop system composed of a reactive neural agent, synthesised from data and implemented by a feed-forward ReLU-activated neural network (ReLU-FFNN), interacting with a non-deterministic environment. Intuitively, the system follows the usual agent-environment loop of observations (of the environment by the agent) and actions (by the agent onto the environment). To model the complexity and partial observability of rich environments, we assume that the neural agent is interacting with a non-deterministic environment, where non-deterministic updates of the environment's state disallow the agent from fully controlling and fully observing the environment's state. Under these assumptions, differently from all related work, the system's evolution is not linear but branching in the future.

We study the verification problem of these systems against a branching time temporal logic. As is known, scalability is a concern in verification and is also an issue in the case of neural systems. To alleviate these difficulties, we are here concerned with a method that is aimed at finding shallow bugs in the system execution, i.e., malfunctions that are realised within a few steps from the system's initialisation. This kind of analysis has been shown to be of particular importance in applications, see, e.g., bounded model checking (BMC) [6], as, experimentally, bugs are often realised after a limited number of steps. Given this, we focus on a bounded version of CTL, i.e., a language expressing temporal properties realisable in a limited number of execution steps. This allows us to reason about applications where the agents ought to bring about a state of affairs within a finite number of steps, or to verify whether a system remains within safety bounds within a number of steps. This enables us to retain decidability even if we consider infinite domains over the reals for the system's state variables, whereas the verification problem for plain CTL is undecidable, as we show. To further alleviate the difficulty of the verification problem, we also introduce a novel algorithm that checks for the occurrence of bugs in parallel over the execution paths. As we show, in the case of bounded safety specifications, this enables us to return a bug to the user as soon as a violation is identified on any of the branching paths that are explored in parallel. This gives considerable advantages in applications, as we show in an avionics application.

A key feature of the parallel verification procedure that we introduce lies in its completeness: we can determine with precision when a potentially infinite set of states (up to a number of steps

Proc.of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 9–13, 2020, Auckland, New Zealand.

from the systems's initialisation) satisfies a temporal formula. While this results in a heavier computational cost than some incomplete approaches, there are obvious benefits in precise verification, notably the lack of false positives and false negatives. To the best of our knowledge this is the first sound and complete verification framework for closed-loop neural systems that accounts for non-deterministic, branching temporal evolutions.

The rest of the paper is organised as follows. After discussing related work, in Section 2 we formally define systems composed by a neural agent, implemented by a ReLU-FFNN, interacting with non-deterministic environments. We analyse the resulting models built on branching executions and define a bounded version of the branching temporal logic CTL to express specifications of these systems. After defining the verification problem, Section 3 introduces monolithic and compositional verification algorithms with a complexity study. In this context we show results ranging from undecidability for unbounded reachability, to coNExpTime upper bound for bounded CTL. We present a toolkit for the practical verification of these systems in Section 4, implementing said procedure, providing additional functionalities, and reporting the experimental results obtained. We conclude in Section 5.

Related Work. In [2] a closed-loop neural agent-environment system was put forward and analysed. Like the present contribution the agent was modelled via a ReLU-FFNN. However, differently from here, a simple deterministic environment was considered. As a consequence, the system executions were linear and only bounded reachability properties were analysed. [1] extended this work to neural agents formalised via recurrent ReLU-activated neural networks and verified the resulting linear system executions against bounded LTL properties. In contrast, the model put forward here can account for complex, partially observable environments resulting in branching traces, and the strictly more expressive specification language allows for existential and universal quantification over paths. In addition, while the papers above focus on sequential verification procedures, we here develop a parallel approach specifically tailored at identifying shallow bugs efficiently. This requires novel verification algorithms and mixed-integer linear programming [38] (MILP) encodings.

A number of other proposals have also addressed the issue of closed loop systems. For example, [19] presents an approach based on hybrid systems to analyse a control-plant where neural networks are synthesised controllers. Their approach is incomparable with the one here pursued, since they target sigmoidal activation functions (while we focus on ReLU activation functions). Also their verification procedure is not complete, while completeness is a key objective here. Similarly, [10, 17, 21, 39] present work addressing closed loop systems with learned controllers and focus on reachable set estimation and, hence, incomplete techniques for such systems.

Lastly, there has been recent activity on complete approaches for verifying standalone ReLU-FFNNs [5, 11, 22, 25, 30, 36]. The systems considered in these approaches are not closed-loop and do not incorporate the environment. This makes the problems considered there different from those analysed here; for instance no temporal evolution can be considered for neural network-controlled agents interacting with an environment.

More broadly, this line of work is related to long standing efforts in BMC [3, 32] that are tailored to finding malfunctions easily accessible from the initial states. While our approach is technically different from BMC, it shares with it the characteristic of being more efficient than full exploration methods when only a fraction of the model needs to be explored.

2 NEURAL AGENT-ENVIRONMENT SYSTEMS

In this section we introduce systems with a neural agent operating on a non-deterministic environment (NANES). These are an extension to non-deterministic environments of the deterministic neural agent-environment systems put forward in [2].

In contrast to traditional models of agency, where the agent's behaviour is given in an agent-based programming language, a NANES accounts for the recent shift to synthesise the agents' behaviour from data [20]; we consider agent protocol functions implemented via feed-forward ReLU neural networks[1] (ReLU-FFNNs) [16]. Differently from [2], following the dynamism and unpredictability of the environments where autonomous agents are typically deployed [27], a NANES models interactions of an agent with a partially observable environment. In this setting an agent cannot observe the full environment state, and therefore cannot deterministically predict the effect of any of its actions.

We now proceed to a formal description of NANES components: a neural agent and a non-deterministic environment. To this end, we fix a set $S \subseteq \mathbb{R}^m$ of environment states and a set $Act \subseteq \mathbb{R}^n$ of actions, for $m, n \in \mathbb{N}$. We assume that the agent is stateless and that its protocol (also known as action policy) has already been synthesised, e.g., via reinforcement learning [34], and is implemented via a ReLU-FFNN or via a piecewise-linear (PWL) combination of them.

Definition 2.1 (Neural Agents). Let S be a set of environment states. A *neural agent* (or simply an *agent*) Ag acting on an environment is defined as the tuple $Ag = (Act, prot)$, where:

- Act is a set of actions;
- $prot : S \to Act$ is a *protocol function* that determines the action the agent will perform given the current state of the environment. Specifically, given ReLU-FFNNs $N_1, \ldots, N_h$ computing functions $f_{N_1}, \ldots, f_{N_h}$, $h \geq 1$, $prot$ is a PWL combination of the latter.

When $h = 1$, $prot(s)$ can be defined, e.g., as $f_{N_1}(s)$ for $s \in S$.

The environment is stateful and non-deterministically updates its state in response to the actions of the agent.

Definition 2.2 (Non-deterministic Environments). An *environment* is a tuple $E = (S, t_E)$, where:

- $S \subseteq \mathbb{R}^m$ is a set of states.
- $t_E : S \times Act \to 2^S$ is a transition function which determines a *finite* set of next possible environment states given its current state and the agent's action.

Given the above we can now define a closed-loop system comprising of an agent interacting with an environment.

Definition 2.3 (NANES). A *Neural Agent operating on a Non-Deterministic Environment System (NANES)* is a tuple $\mathcal{S} = (Ag, E, I)$ where $Ag = (Act, prot)$ is a neural agent, $E = (S, t_E)$ is an environment, and $I \subseteq S$ is a set of initial states for the environment.

[1] Specifically, we consider fully-connected feed-forward neural networks where hidden layers are activated by the widely used Rectified Linear Unit (ReLU) activation function [28], defined as $\mathrm{ReLU}(x) := \max(0, x)$.

Hereafter we assume the environment's transition function is PWL and its set of initial states is expressible as a set of linear constraints over integer and real-valued variables. Note this does not prevent NANES from modelling a wide class of non-linear environments as these can be approximated to arbitrary precision [8].

With each NANES $\mathcal{S}$ we can associate a model $\mathcal{M}_\mathcal{S}$ capturing its evolutions that is used to interpret temporal specifications.

Definition 2.4 (Model). Given a NANES system $\mathcal{S} = (Ag, E, I)$, its associated *temporal model* $\mathcal{M}_\mathcal{S}$ is a pair (R, T) where R is the set of environment states *reachable* from I via T, $I \subseteq R \subseteq S$, and $T \subseteq R \times R$ is the successor relation defined by $(s, s') \in T$ iff $s' \in t_E(s, prot(s))$.

In the rest of the paper, we assume to have fixed a NANES $\mathcal{S}$ and the associated model $\mathcal{M}_\mathcal{S}$. An *$\mathcal{M}_\mathcal{S}$-path*, or simply *path*, is an infinite sequence of states $s_1 s_2 \ldots$ where $s_i \in R$ and s_{i+1} is a successor of s_i, i.e. $(s_i, s_{i+1}) \in T$, for each $i \geq 1$. Given a path ρ we use $\rho(i)$ to denote the i-th state in ρ. For an environment state $s = (a_1, \ldots, a_m)$, we write paths(s) to denote the set of all paths originating from s and we use $s.d$ to denote its d-th component a_d.

We verify NANES against properties expressed in a bounded variant of the temporal logic CTL [7]. Inspired by Real-Time Computation Tree Logic (RTCTL) [12], formulae of bounded CTL build upon temporal modalities indexed with natural numbers denoting the temporal depth up to which the formula is evaluated.

Definition 2.5 (Bounded CTL). Given a set of environment states $S \subseteq \mathbb{R}^m$, the *bounded CTL specification language over linear inequalities*, denoted bCTL$_{\mathbb{R}^<}$, is defined by the following BNF:

$$\begin{array}{lll} \varphi & ::= & \alpha \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid EX^k\varphi \mid AX^k\varphi, \\ \alpha & ::= & c_1(d_1) + \cdots + c_l(d_l)\, op\, c, \end{array}$$

where $op \in \{<, >\}$, $d_i \in \{1, \ldots, m\}$, $c_i, c \in \mathbb{R}$, and $k \in \mathbb{N}$.

Here atomic propositions α are linear constraints on the components of a state. For instance, the atomic proposition $(d_1) + (d_2) < 2$ states that "the sum of the d_1-st and d_2-nd components is less than 2." The temporal formula $EX^k\varphi$ stands for "there is a path such that φ holds after k time steps", whereas $AX^k\varphi$ stands for "in all paths φ holds after k time steps". Moreover, bounded until $E(\varphi U^k \psi)$ ("there is a path such that ψ holds within k time steps, and where φ holds up until then") can be defined by the abbreviations $E(\varphi U^1 \psi) \triangleq \psi \vee (\varphi \wedge EX^1\psi)$, and $E(\varphi U^k \psi) \triangleq \psi \vee (\varphi \wedge EX^1 E(\varphi U^{k-1}\psi))$ for $k > 1$, and analogously with $A(\varphi U^k \psi)$.

Although bCTL$_{\mathbb{R}^<}$ does not include any form of negation, it still allows us to express arbitrary CTL formulae of bounded temporal depth since it supports all Boolean and temporal operators with their duals. Note also that although bCTL$_{\mathbb{R}^<}$ does not support non-strict inequalities, one can in practice remedy this at the expense of completeness through the use of slack variables [38] to create an approximation of a theoretically closed feasble set, which is common practice when modelled using MILP. Consider for instance an atomic proposition of the form $e \geq c$. It can be replaced with the constraints $e > c - \varepsilon$ and $\varepsilon > 0$, where ε is a small slack variable.

We now define the logic CTL built from the atoms of bCTL$_{\mathbb{R}^<}$.

Definition 2.6 (CTL). The *branching-time logic* CTL$_{\mathbb{R}^<}$ is defined by the following BNF:

$$\varphi \;::=\; \alpha \mid \neg\varphi \mid \varphi \vee \varphi \mid AX\varphi \mid AF\varphi \mid E(\varphi U \varphi),$$

where α is an atomic proposition in bCTL$_{\mathbb{R}^<}$.

Comparing bCTL$_{\mathbb{R}^<}$ to CTL$_{\mathbb{R}^<}$, we observe that on the one hand $AX^k\varphi$ and $EX^k\varphi$ are expressible, respectively, as $AX(\cdots(AX\varphi)\cdots)$ and $\neg AX(\cdots(AX\neg\varphi)\cdots)$, where AX is applied k times. On the other hand, CTL$_{\mathbb{R}^<}$ includes the AF ("in all paths eventually") and EU (unbounded until) modalities capable of expressing arbitrary reachability, whereas bCTL$_{\mathbb{R}^<}$ admits bounded specifications only. Note that, while bCTL$_{\mathbb{R}^<}$ is clearly less expressive than CTL$_{\mathbb{R}^<}$, it still captures properties of interest. Notably, *bounded safety* is expressible in bCTL$_{\mathbb{R}^<}$ as $AG^k\, safe \triangleq AX^1 safe \wedge \cdots \wedge AX^k safe$ stating that every state on every path is safe within the first k steps.

We interpret bCTL$_{\mathbb{R}^<}$ formulae on a temporal model as follows.

Definition 2.7 (Satisfaction). For a model $\mathcal{M}_\mathcal{S}$, an environment state s, and a bCTL$_{\mathbb{R}^<}$ formula φ, the *satisfaction* of φ at s in $\mathcal{M}_\mathcal{S}$, denoted $(\mathcal{M}_\mathcal{S}, s) \models \varphi$, or simply $s \models \varphi$ when $\mathcal{M}_\mathcal{S}$ is clear from the context, is inductively defined as follows:

$$\begin{array}{lll} s \models c_1(d_1) + \cdots + c_l(d_l)\, op\, c & \text{iff} & (\sum_{i=1}^{l} c_i \cdot s.d_i)\, op\, c; \\ s \models \varphi \vee \psi & \text{iff} & s \models \varphi \text{ or } s \models \psi; \\ s \models \varphi \wedge \psi & \text{iff} & s \models \varphi \text{ and } s \models \psi; \\ s \models EX^k\varphi & \text{iff} & \text{there is } \rho \in \text{paths}(s) \text{ such that } \rho(k) \models \varphi; \\ s \models AX^k\varphi & \text{iff} & \text{for all } \rho \in \text{paths}(s) \text{ we have } \rho(k) \models \varphi. \end{array}$$

We assume the usual definition of satisfaction for CTL$_{\mathbb{R}^<}$; this can be given as standard by using the atomic case from Definition 2.7.

A specification φ is said to be *satisfied by* $\mathcal{S}$ if $(\mathcal{M}_\mathcal{S}, s) \models \varphi$ for all initial states $s \in I$. We denote this by $\mathcal{S} \models \varphi$. It follows that, for example, to check bounded safety we need to verify that from all (possibly infinitely many) initial states no state (out of possibly infinitely many) within the first k evolutions is an unsafe state. This is the basis of the verification problem that we define below.

Definition 2.8 (Verification problem). Given a NANES $\mathcal{S}$ and a formula φ, determine whether $\mathcal{S} \models \varphi$.

In the next section we study the decidability and complexity of the verification problem here introduced.

3 THE VERIFICATION PROBLEM

In this section we study the verification problem for a NANES against CTL and bCTL$_{\mathbb{R}^<}$ specifications. First, we show that verifying against CTL formulae is undecidable, already for deterministic environments and simple reachability properties. In the rest of the section, we focus on bounded CTL, where we develop a decision procedure for the verification problem based on producing a single MILP and checking its feasibility. Then we devise a parallelisable version of the procedure that produces multiple MILPs and that can be particularly efficient at finding counter-examples for bounded safety properties. Following this, we analyse the computational complexity of the verification problem against bCTL$_{\mathbb{R}^<}$ formulae.

3.1 Unbounded CTL

In this subsection we show undecidability of the verification problem for deterministic NANES against simple reachability properties, where a deterministic NANES is a tuple $(Ag = (Act, prot), E = (S, t_E), I)$, where $|t_E(s, a)| = 1$ for all $s \in S$ and $a \in Act$. The undecidability result for arbitrary NANES and full CTL follows.

Theorem 3.1. *Verifying deterministic NANES against formulae of the form $AF\alpha$ is undecidable.*

Proof Sketch. We can show the result by reduction from the Halting problem of a deterministic Turing machine (DTM) M on an input string ω_0, whose tape alphabet consists of symbols 0, 1 and 2, with one halting state (the accepting state).

The idea of the reduction is to construct a NANES $\mathcal{S} = (Ag, E, I)$ such that each state of $\mathcal{S}$ encodes the current configuration of the DTM, i.e., the current state of M, the symbol under the head, and the contents of the tape to the left and right of the head as two real numbers (the former one is read from right to left). I consists of a single state and encodes q_0 (the initial state of M) and ω_0. The run of M on its input can be simulated by appropriately updating the state using the environment transition function (while the agent does not need to do anything). Conversely, it is possible to shift all the logic to the agent's protocol function with a trivial environment.

Finally, we verify $\mathcal{S}$ against the reachability specification φ of the form *AF accept*, where *accept* encodes that M is in the accepting state. Then $\mathcal{S} \models \varphi$ iff M halts on ω_0. It can also be checked that the required environment transition function and the agent's protocol function can be implemented as piecewise-linear functions. □

We observe that the above result holds even for strongly restricted NANES where either the protocol or the transition function is linear (but not both at the same time). As a corollary, we obtain undecidability of the verification problem against full CTL.

Corollary 3.2. *Verifying NANES against CTL$_{\mathbb{R}^<}$ formulae is undecidable.*

3.2 Bounded CTL

We now proceed to investigate the verification problem for the bounded CTL specification language. We start by showing an auxiliary result that allows us to assume without loss of generality that the cardinality of $t_E(s, a)$ is the same for each state s and action a.

Lemma 3.3. *Given a NANES $\mathcal{S} = ((Act, prot), (S, t_E), I)$ and specification $\varphi \in bCTL_{\mathbb{R}^<}$, there is a NANES $\mathcal{S}' = ((Act, prot'), (S', t'_E), I')$, such that $|t'_E(s_1, a_1)| = |t'_E(s_2, a_2)|$ for all $s_1, s_2 \in S'$, and $a_1, a_2 \in Act$, and a specification $\varphi' \in bCTL_{\mathbb{R}^<}$ such that $\mathcal{S} \models \varphi$ iff $\mathcal{S}' \models \varphi'$.*

Proof Sketch. Consider $b = \max_{s \in S, a \in Act} |t_E(s, a)|$. Define the components of $\mathcal{S}'$ such that $|t'_E(s, a)| = b$ for all $s \in S'$, $a \in Act$, and $\mathcal{S} \models \varphi$ iff $\mathcal{S}' \models \varphi'$. S' and I' are defined by $S' = S \times \{0, 1\}$ and $I' = I \times \{1\}$. The added dimension indicates whether a state is valid (1) or not (0). The agent's protocol function $prot'$ is defined as $prot'((s, f)) = prot(s)$ for each $s \in S$, $f \in \{0, 1\}$. The transition function $t'_E((s, f), a)$ returns $t_E(s, a) \times \{1\} \cup \{(s_1, 0), \ldots, (s_{b-l}, 0)\}$ where $|t_E(s, a)| = l$ and $s_1, \ldots, s_{b-l}$ are pairwise distinct states from S. The formula φ' is a copy of φ with atomic propositions α replaced with $\alpha \wedge ((m + 1) > 0.9)$, where $S = \mathbb{R}^m$. □

In the rest of this section we assume that $|t_E(s, a)| = b$ for all s and a, and that t_E is given as b piecewise-linear (PWL) functions $t_i : \mathbb{R}^{m+n} \to \mathbb{R}^m$. Note that this assumption is used when devising the verification procedure presented below.

The procedure that we put forward recasts the verification problem to MILP. It is well known that a PWL function can be MILP-encoded using the "Big-M" method [14]. For instance, the pairs (x, y), where $y = \text{ReLU}(x)$ and $x \in [l, u]$ can be found as solutions to the following set of MILP constraints that use the binary variable δ, real-valued variables x and y and constants l and u:

$$y \geq 0, \quad y \geq x, \quad y \leq u \cdot \delta, \quad y \leq x - l \cdot (1 - \delta)$$

Here, when $\delta = 1$, the constraints imply that $y = x$ and $x \geq 0$, and when $\delta = 0$, the constraints imply that $y = 0$ and $x \leq 0$. Since the function computed by a ReLU-activated neural network can be obtained via successive compositions of the ReLU function and linear transformations, its MILP encoding can be obtained via the composition of constraints of the above form with appropriate linear constraints. The resulting overall MILP is of linear size in the size of the network. Further details of the Big-M encoding of the ReLU function can be found in [25, 36].

Given a MILP program π, we use $vars(\pi)$ to denote the set of variables in π. Denote by $\mathfrak{a}$ the *assignment function* $\mathfrak{a} : vars(\pi) \to \mathbb{R}$, which defines the specific (binary, integer or real) value assigned to a MILP program variable. We write $\mathfrak{a} \models \pi$ if $\mathfrak{a}$ *satisfies* π, i.e., if $\mathfrak{a}(\delta) \in \{0, 1\}$ for each binary variable δ, $\mathfrak{a}(\iota) \in \mathbb{N}$ for each integer variable ι, and all constraints in π are satisfied. Hereafter, we will denote by boldface font tuples of MILP variables (of length m for $S \subseteq \mathbb{R}^m$ the set of environment states) representing an environment state and call them *state variables*.

Monolithic Encoding. We now give a recursive encoding of the verification problem into a single MILP. As a stepping stone, we first encode the computation of a successor environment state as a composition of the protocol function *prot* and of the transition functions t_i. By assumption, *prot* and each t_i is a PWL function, and so the predicate $\boldsymbol{y} = t_i(\boldsymbol{x}, prot(\boldsymbol{x}))$ is expressible as a set of MILP constraints by means of the Big-M method, which we denote by $C_i(\boldsymbol{x}, \boldsymbol{y})$ (note that $\boldsymbol{x} \cup \boldsymbol{y} \subset vars(C_i(\boldsymbol{x}, \boldsymbol{y}))$). Solutions of $C_i(\boldsymbol{x}, \boldsymbol{y})$ represent pairs of consecutive environment states [2]:

Lemma 3.4. *Let $C_i(\boldsymbol{x}, \boldsymbol{y})$ be a MILP program corresponding to $\boldsymbol{y} = t_i(\boldsymbol{x}, prot(\boldsymbol{x}))$. Given two states s and s' in $\mathcal{M}_{\mathcal{S}}$, we have that $s' = t_i(s, prot(s))$ iff there is an assignment $\mathfrak{a}$ to $vars(C_i(\boldsymbol{x}, \boldsymbol{y}))$ such that $s = \mathfrak{a}(\boldsymbol{x})$, $s' = \mathfrak{a}(\boldsymbol{y})$, and $\mathfrak{a} \models C_i(\boldsymbol{x}, \boldsymbol{y})$.*

Denote by bCTL$_{\mathbb{R}^{\leq}}$ the bounded CTL language over atomic propositions α where $op \in \{\leq, \geq\}$ (i.e., linear constraints over non-strict inequalities). As a second step, given a NANES $\mathcal{S}$ and a formula $\varphi \in$ bCTL$_{\mathbb{R}^{\leq}}$, we construct a MILP program $\pi_{\mathcal{S},\varphi}$, whose feasibility corresponds to the existence of a state in $\mathcal{M}_{\mathcal{S}}$ that satisfies φ. For ease of presentation, and without loss of generality, we assume that φ may contain only the temporal modalities EX^1 and AX^1, for which we write EX and AX, respectively[2]. We make use of the *indicator constraints* of the form $(\delta = v) \Rightarrow c$, for a binary variable δ, binary value $v \in \{0, 1\}$ and a linear constraint c, meaning that whenever the value of δ is v, the constraint c should hold. In particular, indicator constraints can be used to naturally express disjunctive cases. For instance, the disjunction $x = 3 \vee x = 5$ can be encoded using two auxiliary binary variables δ_i, $i \in \{1, 2\}$, and the following set of MILP constraints:

$$(\delta_1 = 1) \Rightarrow x = 3, \quad (\delta_2 = 1) \Rightarrow x = 5, \quad \delta_1 + \delta_2 = 1.$$

Here, the constraint $\delta_1 + \delta_2 = 1$ ensures that at least one of the two clauses is satisfied (note that, in general, the above encoding does

[2]For $Q \in \{A, E\}$, the formula $QX^k\varphi$ is equivalent to $QX^1(\cdots(QX^1\varphi)\cdots)$ where QX^1 is applied k times to φ and which grows linearly in k assuming unary encoding of numbers.

$$\pi_{\mathcal{S},\alpha}(\boldsymbol{x}) = \{C_\alpha(\boldsymbol{x})\}, \quad \text{where } C_\alpha(\boldsymbol{x}) \text{ is defined as } c_1 x_{d_1} + \cdots + c_l x_{d_l} \; op \; c \text{ for } \alpha = c_1(d_1) + \cdots + c_l(d_l) \; op \; c \text{ and } \boldsymbol{x} = (x_1, \ldots, x_m),$$

$$\pi_{\mathcal{S},\varphi_1 \vee \varphi_2}(\boldsymbol{x}) = (\delta = 1) \Rightarrow (\pi_{\mathcal{S},\varphi_1}(\boldsymbol{x_1}) \cup \{\boldsymbol{x} = \boldsymbol{x}_1\}) \;\cup\; (\delta = 0) \Rightarrow (\pi_{\mathcal{S},\varphi_2}(\boldsymbol{x_2}) \cup \{\boldsymbol{x} = \boldsymbol{x}_2\}),$$

$$\pi_{\mathcal{S},\varphi_1 \wedge \varphi_2}(\boldsymbol{x}) = \pi_{\mathcal{S},\varphi_1}(\boldsymbol{x}) \cup \pi_{\mathcal{S},\varphi_2}(\boldsymbol{x}),$$

$$\pi_{\mathcal{S},EX\varphi}(\boldsymbol{x}) = \bigcup_{i=1}^{b} (\delta_i = 1) \Rightarrow (C_i(\boldsymbol{x}, \boldsymbol{y}_i) \cup \{\boldsymbol{y} = \boldsymbol{y}_i\}) \;\cup\; \{\delta_1 + \cdots + \delta_b = 1\} \;\cup\; \pi_\varphi(\boldsymbol{y}),$$

$$\pi_{\mathcal{S},AX\varphi}(\boldsymbol{x}) = C_1(\boldsymbol{x}, \boldsymbol{y}_1) \cup \cdots \cup C_b(\boldsymbol{x}, \boldsymbol{y}_b) \;\cup\; \pi_{\mathcal{S},\varphi}(\boldsymbol{y}_1) \cup \cdots \cup \pi_{\mathcal{S},\varphi}(\boldsymbol{y}_b),$$

where the binary variables $\delta, \delta_1, \ldots, \delta_b$, the state variables $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_b, \boldsymbol{y}$, and all auxiliary variables in $C_i(\boldsymbol{x}, \boldsymbol{y}_i)$ are fresh.

Figure 1: Monolithic encoding $\pi_{\mathcal{S},\varphi}$ for $\varphi \in$ bCTL$_{\mathbb{R}^{\leq}}$.

not rule out an assignment where both clauses are satisfied at the same time). Given a binary variable δ and a set of constraints π, we hereafter abbreviate $\{(\delta = v) \Rightarrow c \mid c \in \pi\}$ to $(\delta = v) \Rightarrow \pi$.

We now define the *monolithic* encoding $\pi_{\mathcal{S},\varphi}$.

Definition 3.5. Given a NANES $\mathcal{S}$ and a formula $\varphi \in$ bCTL$_{\mathbb{R}^{\leq}}$, their *monolithic MILP encoding* $\pi_{\mathcal{S},\varphi}$ is defined as the MILP program $\pi_{\mathcal{S},\varphi}(\boldsymbol{x})$, where $\boldsymbol{x}$ is a tuple of fresh state variables, and $\pi_{\mathcal{S},\varphi}(\boldsymbol{x})$ is built inductively using the rules in Figure 1.

In the encoding in Figure 1, the base case $\pi_{\mathcal{S},\alpha}(\boldsymbol{x})$ for an atom α produces the MILP program consisting of a single linear constraint corresponding to α and using variables in $\boldsymbol{x}$. Each inductive case depends on the state variables $\boldsymbol{x}$ but might in turn generate programs for subformulas which depend on freshly created state variables different to $\boldsymbol{x}$ (such as $\boldsymbol{x}_1$, $\boldsymbol{x}_2$, $\boldsymbol{y}$, etc). All other auxiliary variables employed in the encoding are also fresh, preventing undesirable interactions between unrelated branches of the program.

- Disjunctions use a binary variable δ and two sets of indicator constraints. In a feasible assignment $\mathfrak{a}$, when δ is 1, φ_1 is satisfied and the variables $\boldsymbol{x}$ take the values of the variables $\boldsymbol{x}_1$, while when δ is 0, φ_2 is satisfied and $\boldsymbol{x}$ takes the values of $\boldsymbol{x}_2$.
- We encode conjunction as the union of the constraints for each of the conjuncts, which all must be satisfied at the same time.
- We encode EX via b-ary disjunction: there are b possible next states and each disjunct chooses one of them by ensuring that the relevant $C_i(\boldsymbol{x}, \boldsymbol{y}_i)$ is satisfied. The variables for the successor state $\boldsymbol{y}$ are assigned accordingly to this choice; moreover, the subprogram for φ depends on them. Notably, only one copy of $\pi_{\mathcal{S},\varphi}$ is required.
- To satisfy $AX\varphi$, all b possible successor states should satisfy φ, and so we take the union of all $C_i(\boldsymbol{x}, \boldsymbol{y}_i)$ and of b copies of $\pi_{\mathcal{S},\varphi}$, each depending on one of the successor state variables $\boldsymbol{y}_i$.

Note that the size of $\pi_{\mathcal{S},\varphi}$ may grow exponentially due to b repetitions of $\pi_{\mathcal{S},\psi}$ in $\pi_{\mathcal{S},AX\psi}(\boldsymbol{x})$; for $\varphi = AX^k\alpha$, the size of $\pi_{\mathcal{S},\varphi}$ is $O(k \cdot b^k \cdot |\mathcal{S}|)$. The same estimate works in the general case for the temporal bound k of φ. On the other hand, when φ contains no AX operator, the size of $\pi_{\mathcal{S},\varphi}$ remains polynomial $O(k \cdot b \cdot |\mathcal{S}|)$.

We can prove that $\pi_{\mathcal{S},\varphi}$ is as intended.

Lemma 3.6. *Given a NANES $\mathcal{S}$ and a formula $\varphi \in bCTL_{\mathbb{R}^{\leq}}$, the following are equivalent:*

(1) There exists a state s in $\mathcal{M}_\mathcal{S}$ such that $s \models \varphi$.
(2) There exists an assignment $\mathfrak{a}$ to vars$(\pi_{\mathcal{S},\varphi}(\boldsymbol{x}))$ such that $\mathfrak{a} \models \pi_{\mathcal{S},\varphi}(\boldsymbol{x})$ and $s = \mathfrak{a}(\boldsymbol{x})$.

Finally, we can exploit Lemma 3.6 to devise a procedure that solves the verification problem by restricting $\boldsymbol{x}$ to the initial states of $\mathcal{S}$. The procedure is given by Algorithm 1. Its soundness and completeness is shown by the following.

Algorithm 1 The monolithic MILP verification procedure.

```
procedure MONO-VERIFY(S, φ)
    Input: NANES S = (Ag, E, I); formula φ ∈ bCTL_{R<}
    Output: True/False
    φ_I ← Boolean formula representing I
    φ' ← NNF(¬φ ∧ φ_I)
    π_{S,φ'} ← MILP associated with S and φ'
    feasible ← MILP_SOLVER(π_{S,φ'})
    return ¬feasible
```

Theorem 3.7. *Given a NANES $\mathcal{S}$ and a formula $\varphi \in bCTL_{\mathbb{R}^{<}}$, Algorithm 1 returns* False *iff $\mathcal{S} \not\models \varphi$.*

Proof. Suppose that Algorithm 1 returns *False*. It follows that $\pi_{\mathcal{S},\neg\varphi\wedge\varphi_I}(\boldsymbol{x})$ is feasible. So, by Lemma 3.6, there exists an assignment $\mathfrak{a}$ to vars$(\pi_{\mathcal{S},\neg\varphi\wedge\varphi_I}(\boldsymbol{x}))$ such that $\mathfrak{a} \models \pi_{\mathcal{S},\neg\varphi\wedge\varphi_I}(\boldsymbol{x})$. Moreover, for $s = \mathfrak{a}(\boldsymbol{x})$, we have that $s \models \varphi_I$ and $s \models \neg\varphi$. It follows that $s \in I$ and $s \not\models \varphi$, and consequently, $\mathcal{S} \not\models \varphi$. Conversely, if there exists $s \in I$ such that $s \not\models \varphi$, we obtain that there is an assignment satisfying $\pi_{\mathcal{S},\neg\varphi\wedge\varphi_I}(\boldsymbol{x})$, and therefore Algorithm 1 returns *False*. □

Recall that strict inequalities are not supported in the MILP solver. Therefore note that we only pass $\neg\varphi$ (the negation of the specification), and the initial state φ_I (expressed only in terms of non-strict intequalities) to the MILP solver in *negation normal form* (NNF). In this process, negation is eliminated by pushing it down and through the atoms resulting in all strict inequalities of atoms of the original specification φ being converted to non-strict inequalities.

Compositional Encoding. Observe that due to its handling of disjunctions, the previously introduced encoding $\pi_{\mathcal{S},\varphi}$ might result in excessively large programs whose feasibility is a computationally expensive task. We now propose a different encoding that instead of delegating disjunction to the MILP solver (the $\varphi_1 \vee \varphi_2$ and $EX\varphi$ cases) creates a separate program for each disjunct. More specifically, for a formula φ, we define a set $\Pi_{\mathcal{S},\varphi}$ of MILP programs with the property that there exists a state s in $\mathcal{M}_\mathcal{S}$ such that $s \models \varphi$ iff at least one of the programs in $\Pi_{\mathcal{S},\varphi}$ is feasible. A specific feature of this encoding lies in its parallelisability. Due to this, it is particularly amenable to efficiently finding bugs that can be reached within a few steps along some of the paths from the initial states, similarly to BMC [6]. We demonstrate this experimentally in the next section after having introduced the encoding here.

$$
\begin{aligned}
\Pi_{\mathcal{S},\alpha}(\boldsymbol{x}) &= \{[C_\alpha(\boldsymbol{x})]\},\\
\Pi_{\mathcal{S},\varphi_1\vee\varphi_2}(\boldsymbol{x}) &= \Pi_{\mathcal{S},\varphi_1}(\boldsymbol{x}) \cup \Pi_{\mathcal{S},\varphi_2}(\boldsymbol{x}),\\
\Pi_{\mathcal{S},\varphi_1\wedge\varphi_2}(\boldsymbol{x}) &= \Pi_{\mathcal{S},\varphi_1}(\boldsymbol{x}) \times \Pi_{\mathcal{S},\varphi_2}(\boldsymbol{x}),\\
\Pi_{\mathcal{S},EX\varphi}(\boldsymbol{x}) &= \bigcup_{i=1}^{b} \{[C_i(\boldsymbol{x},\boldsymbol{y})]\} \times \Pi_{\mathcal{S},\varphi}(\boldsymbol{y}),\\
&\quad \text{where the state variables } \boldsymbol{y} \text{ and all remaining variables in } C_i(\boldsymbol{x},\boldsymbol{y}) \text{ are fresh,}\\
\Pi_{\mathcal{S},AX\varphi}(\boldsymbol{x}) &= \bigtimes_{i=1}^{b} \{[C_i(\boldsymbol{x},\boldsymbol{y}_i)]\} \times \Pi_{\mathcal{S},\varphi}(\boldsymbol{y}_i),\\
&\quad \text{where the state variables } \boldsymbol{y}_1,\ldots,\boldsymbol{y}_b \text{ and all remaining variables in } C_i(\boldsymbol{x},\boldsymbol{y}_i) \text{ are fresh.}
\end{aligned}
$$

Figure 2: Compositional encoding $\Pi_{\mathcal{S},\varphi}$ for $\varphi \in$ bCTL$_{\mathbb{R}^{\leq}}$.

Below, given a set C of linear constraints, we write $[C]$ to denote the respective MILP program. Given sets $A = \{[A_1],\ldots,[A_p]\}$ and $B = \{[B_1],\ldots,[B_q]\}$ of MILP programs, we write $A{\times}B$ to denote the *product* of A and B computed as $\{[A_i \cup B_j] \mid i = 1,..,p,\ j = 1,..,q\}$.

Definition 3.8. Given a NANES $\mathcal{S}$ and a formula $\varphi \in$ bCTL$_{\mathbb{R}^{\leq}}$, their *compositional MILP encoding* $\Pi_{\mathcal{S},\varphi}$ is defined as the set of MILP programs $\Pi_{\mathcal{S},\varphi}(\boldsymbol{x})$, where $\boldsymbol{x}$ is a tuple of fresh state variables, and $\Pi_{\mathcal{S},\varphi}(\boldsymbol{x})$ is built inductively using the rules in Figure 2.

Following the monolithic encoding in Figure 1, in Figure 2 $C_\alpha(\boldsymbol{x})$ is the linear constraint corresponding to the atomic proposition α defined over $\boldsymbol{x}$. We use the same convention regarding the state and auxiliary variables of subprograms. In $\Pi_{\mathcal{S},\varphi}$ every program π represents one of the encodings of φ.

- For disjunction we take the union of the two sets of encodings.
- Every encoding of $\varphi_1 \wedge \varphi_2$ consists of an encoding of φ_1 and of an encoding of φ_2, therefore we take the product of the two sets.
- Every encoding of $EX\varphi$ is an encoding of φ extended with the constraints $C_i(\boldsymbol{x},\boldsymbol{y})$ for a single i.
- Every encoding of $AX\varphi$ consists of b (possibly different) encodings of φ extended with the constraints $C_i(\boldsymbol{x},\boldsymbol{y}_i)$ for $i = 1,\ldots,b$.

The set $\Pi_{\mathcal{S},\varphi}$ grows exponentially with the temporal depth of φ; however each program in the set can be smaller than the monolithic MILP $\pi_{\mathcal{S},\varphi}$.

Similarly to Lemma 3.6 we can prove that $\Pi_{\mathcal{S},\varphi}$ is as intended.

Lemma 3.9. *Given a NANES $\mathcal{S}$ and a formula $\varphi \in$ bCTL$_{\mathbb{R}^{\leq}}$, the following are equivalent:*

(1) There exists a state s in $\mathcal{M}_{\mathcal{S}}$ such that $s \models \varphi$.
(2) There is a MILP $\pi(\boldsymbol{x}) \in \Pi_{\mathcal{S},\varphi}(\boldsymbol{x})$ and an assignment $\mathfrak{a}$ to vars$(\pi(\boldsymbol{x}))$ such that $s = \mathfrak{a}(\boldsymbol{x})$ and $\mathfrak{a} \models \pi(\boldsymbol{x})$.

Based on Lemma 3.9, we can devise a verification procedure that searches for a feasible MILP in the set of MILPs generated by the encoding of Figure 2. This procedure is presented in Algorithm 2. Importantly, it naturally lends itself to parallelisation when checking feasibility of the generated programs. In turn this enables us to check in parallel for a possible falsification of formulas in which the temporal operator is universally quantified as in $AX^k\alpha$. As we will see in the next section, this will become particularly useful when verifying bounded safety.

Computational Complexity. Finally we study the complexity of the verification problem for bCTL$_{\mathbb{R}^{<}}$. The upper bound follows from the monolithic verification procedure and the lower bound can be obtained by reduction from the validity problem of QBF.

Algorithm 2 The compositional MILP verification procedure.

```
1:  procedure COMP-VERIFY(S, φ)
2:      Input: NANES S = (Ag, E, I); formula φ ∈ bCTL_{R<}
3:      Output: True/False
4:      feasible ← False
5:      φ_I ← Boolean formula representing I
6:      φ' ← NNF(¬φ ∧ φ_I)
7:      Π_{S,φ'} ← Set of MILPs associated with S and φ'
8:      for π in Π_{S,φ'} do
9:          aux ← MILP_SOLVER(π)
10:         if aux is True then
11:             feasible ← True
12:             break
13:     return ¬feasible
```

Theorem 3.10. *Verifying NANES against bCTL$_{\mathbb{R}^{<}}$ is in coNExpTime and PSpace-hard in combined complexity.*

We also show that the complexity of the verification problem is reduced to coNP for the bounded safety fragment of bCTL$_{\mathbb{R}^{<}}$.

Corollary 3.11. *Verifying NANES against bounded safety properties is coNP-complete in combined complexity.*

Proof Sketch. The upper bound follows from the fact that we can check whether a property $\varphi = AG^k\,safe$ is not satisfied by $\mathcal{S}$ by guessing an initial state s and a path ρ of length k originating from s, and by verifying that $\rho(i) \not\models safe$ for some $i = 1,\ldots,k$. If such an initial state s exists, then there exists an initial state s' with the same properties of polynomial size. This follows from the encoding into MILP and the fact that if a MILP instance is feasible, there is a solution of polynomial size. The lower bound can be adapted from the NP lower bound of the satisfiability problem of neural networks properties [22], and holds already for one-step formulae. □

4 IMPLEMENTATION AND EXPERIMENTS

We have implemented the verification procedures described in the previous section in an open source toolkit called NANESVerify [29]. The tool takes as input a bCTL$_{\mathbb{R}^{<}}$ specification φ and a NANES $\mathcal{S}$ in the form of ReLU-FFNNs implementing the agent, piecewise linear (PWL) functions (possibly given as ReLU-FFNNs) implementing the environment and a set I of the initial states in the form of a hyper-rectangle which can be encoded as $x_1 \geq l_1 \wedge x_1 \leq u_1 \wedge \cdots \wedge x_m \geq l_m \wedge x_m \leq u_m$ for hyper-rectangle $[l_1,u_1]\times\cdots\times[l_m,u_m]$ and state variables $\boldsymbol{x} = (x_1,\ldots,x_m)$. The top-level call to the tool returns `True` if φ is satisfied on $\mathcal{S}$, and returns `False` if φ fails for some initial state of $\mathcal{S}$. In the latter case, a trace in the form of state-action pairs is produced, giving an example run of the system which failed to satisfy the specification.

The user can specify a parameter to determine whether the monolithic or compositional procedure with parallel or sequential execution is to be used. When using sequential execution, NANESVerify follows the respective procedures from Algorithms 1 and 2. For the compositional procedure with parallel execution, NANESVerify

performs the computation in line 9 of Algorithm 2 asynchronously across eight worker processes running a separate Gurobi instance for each MILP. The main process finishes either when a MILP-solving job terminates with a feasible solution (finding a counter-example), all jobs gave infeasible results, or no result was returned within a given time limit.

We used Python to implement the tool and relied on Gurobi ver. 8.1 [15] as a back-end to resolve the feasibility of the generated MILP problems. When constructing the Big-M encodings of the neural networks, the lower and upper bounds for each neuron are determined using symbolic linear relaxation [37] starting from the bounds of the input nodes given by I. For other MILP variables encountered, we propagate their bounds throughout the encoding using interval arithmetic. For the compositional encoding, we delegate disjunctions at the level of atomic propositions to the MILP solver, which avoids the unnecessary blow-up of the number of MILPs generated and can still be efficiently handled by the solver.

Aircraft Collision Avoidance System Example. To validate the toolkit we use a scenario involving two aircraft, the *ownship* and the *intruder*, where the ownship is equipped with a collision avoidance system referred to as VerticalCAS [21]. The intruder is assumed to follow a constant horizontal trajectory. VerticalCAS once every second issues vertical climbrate advisories to the ownship pilot, to avoid a *near mid-air collision (NMAC)*, a region where the ownship and intruder are separated by less than 100ft vertically and 500ft horizontally. The possible advisories are:

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.

The advisories instruct the pilot to accelerate until the vertical climbrate of the ownship complies with the advisory. For some advisories, e.g. DND, the pilot can choose any acceleration in $[g/4, g/3]$, where g represents the gravitational constant 32.2 ft/s^2.

We hereafter denote by $[m]$ the set $\{1, .., m\}$. The set of tuples $S = (h, \dot{h}_0, \tau, \text{adv}) \in [-3000, 3000] \times [-2500, 2500] \times [0, 40] \times [9]$ describe an ownship–intruder *encounter*, where:

1) h (ft): Intruder altitude relative to ownship.
2) $\dot{h}_0$ (ft/s): Ownship vertical climbrate.
3) τ (s): Time to loss of horizontal separation.
4) adv: The previous advisory issued by VerticalCAS.

The vertical geometry of the encounter is given by h and $\dot{h}_0$, and τ reports the seconds until the ownship (black) and intruder (red) are no longer horizontally separated, illustrated in Fig. 3.

The VerticalCAS system is composed of nine ReLU-FFNNs $F = \{(f_{N_i} : \mathbb{R}^3 \to \mathbb{R}^9) : i \in [9]\}$, one for each 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.

NANES Encoding. We model VerticalCAS as a neural agent with protocol function $prot(s) = \arg\max(\text{apply}(\text{select}(s), s))$ on input

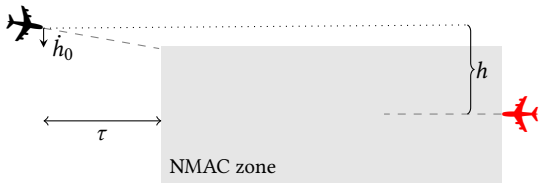

Figure 3: VerticalCAS encounter geometry

state $s = (h, \dot{h}_0, \tau, \text{adv}) \in S$, producing an action $a \in Act = [9]$ corresponding to the highest-scoring advisory, where:

- select: $S \to F$ selects the neural network corresponding to the previous advisory adv, defined $\text{select}(s) = f_{\text{adv}}$,
- apply: $F \times \mathbb{R}^4 \to \mathbb{R}^9$ computes the output of a neural network given a state, defined as $\text{apply}(f, s) = f(h, \dot{h}_0, \tau)$,
- $\arg\max$: $\mathbb{R}^9 \to [9]$ returns the index of the score with highest value from a neural network's output.

Since each of the above functions and the ReLU-FFNNs are PWL, the composition *prot* is also PWL.

We model the ownship pilot's non-deterministic behaviour in the *environment* of $\mathcal{S}$. Thus, the environment transition function t_E "chooses" an acceleration and determines the next state of the environment through the state transition dynamics. As described in [21], the acceleration chosen by the pilot is assumed to be from a continuous interval, but we bound the number of possible successor states of t_E, by discretising the set of possible accelerations into b equally spaced cells. Here we choose $b = 3$. Take for example advisory DND; the set of next possible accelerations are $\{g/4, 7g/24, g/3\}$.

Assume a boolean predicate compliant: $S \times Act \to \mathbb{B}$ which returns `True` iff the current vertical climbrate of the ownship is compliant with the advisory issued by the agent. Non-zero accelerations are chosen only if compliant does not hold, otherwise the pilot maintains a constant climbrate, i.e., $\ddot{h}_0^{(i)} = 0$ for $i \in [b]$. Given the current state $s \in S$, the issued advisory $\text{adv}' = prot(s)$, and the set of b accelerations $\{\ddot{h}_0^{(i)} : i \in [b]\}$ corresponding to the advisory adv′, we define each of the transition functions $t_1, \ldots, t_b$ for t_E as:

$$t_i\left(\begin{bmatrix} h \\ \dot{h}_0 \\ \tau \\ \text{adv} \end{bmatrix}, \text{adv}'\right) = \begin{bmatrix} h - \dot{h}_0 \Delta\tau - 0.5\ddot{h}_0^{(i)}\Delta\tau^2 \\ \dot{h}_0 + \ddot{h}_0^{(i)}\Delta\tau \\ \tau - \Delta\tau \\ \text{adv}' \end{bmatrix},$$

where $\Delta\tau = 1$ and $i \in [b]$.

Experimental Results. We tested NANESVerify on the following safety specification:

$$\varphi^k = AX^k\,((1) > 100 \vee (1) < -100)$$

for various values of k. The formula φ^k is satisfied if from every initial state in I, all possible evolutions of the system remain *safe* after k time steps, i.e., there does not exist a state in I which, after k time steps, can lead to the ownship entering the unsafe region ($|h| \leq 100$), which may potentially lead to an NMAC for small values of τ (recall that in bCTL$_{\mathbb{R}^<}$, the term (1) represents the first component of the state s and so refers to $s.1 = h$). We consider the verification problem with the set of initial states $I = [-133, -129] \times \{-19.5, -22.5, -25.5, -28.5\} \times \{25\} \times \{\text{COC}\}$. This

	Comp-Par				Comp-Seq						Monolithic					
k	−19.5	−22.5	−25.5	−28.5	−19.5	−22.5	−25.5	−28.5	C	V	−19.5	−22.5	−25.5	−28.5	C	V
1	0.629s	0.608s	0.649s	0.652s	0.728s	0.819s	0.737s	0.750s	3037	2993	0.039s	0.039s	0.041s	0.042s	3190	2996
2	2.901s	2.730s	1.092s	1.429s	5.392s	5.594s	0.623s	0.618s	6668	5981	4.399s	6.450s	1.444s	3.323s	6974	5987
3	10.67s	1.716s	1.918s	1.824s	26.06s	0.986s	0.961s	0.964s	10300	8969	23.33s	14.58s	12.79s	13.59s	10759	8978
4	39.58s	40.91s	2.474s	2.570s	109.9s	108.7s	1.404s	1.417s	13930	11957	–	–	377.5s	29.96s	14542	11969
5	145.6s	156.3s	159.8s	3.830s	433.5s	481.2s	512.4s	2.244s	17560	14945	–	–	–	751.1s	18325	14960
6	797.4s	544.8s	573.5s	568.8s	2174.s	1639.s	1826.s	1859.s	21198	17933	–	–	–	–	22116	17951

Table 1: Verification times for a VerticalCAS system against the property φ^k for different values of k and $\dot{h}_0$. Greyed-out cells indicate a `False` result, otherwise a `True` result. We use dashes '–' to indicate a two hour timeout. The C and V columns indicate the number of MILP constraints and variables, respectively.

is a potentially risky encounter with the intruder initially below the ownship, but with the ownship descending towards the intruder.

All results were obtained on a machine with an Intel Core i7-6700 3.40GHz CPU with 16GB of RAM, running a 64-bit version of Ubuntu 16.04. The results for the monolithic procedure are denoted Monolithic, and the results for the compositional procedure with parallel and sequential execution are denoted Comp-Par and Comp-Seq, respectively. In Table 1, we report the performance of the tool in terms of the amount of time (in seconds) to resolve the specification φ^k for $k \in \{1, \ldots, 6\}$ with initial climbrates $\dot{h}_0 \in \{-19.5, -22.5, -25.5, -28.5\}$ for each of the execution modes. For all cases we use a fixed timeout of two hours. We also report the size (number of constraints and variables) of the single MILP problem $\pi_{\mathcal{S},\varphi^k}$ when using Monolithic and for the compositional cases, the size of the largest MILP problem constructed for $\Pi_{\mathcal{S},\varphi^k}$.

In Table 1 we see a climbrate of −28.5 ft/s resulting in a period where the ownship enters the unsafe region for four time steps. For smaller descent rates, the time spent in the unsafe region decreases, until for $\dot{h}_0 = -19.5$ where the ownship remains safe for the entire period. Upon analysing the trace produced by NANESVerify for $(\dot{h}_0, k) = (-22.5, 3)$, the agent produces advisory CL1500 at each time step, causing the pilot to accelerate at $g/4$ ft/s^2 in an attempt to climb to avoid colliding with the intruder. The descent rate was not reduced quickly enough to avoid the unsafe state $(h, \dot{h}_0, \tau, \text{adv}) = (-97.719,\ 1.65,\ 22,\ \text{CL1500})$ being reached by the third timestep.

Overall, Comp-Par is the most performant method for resolving the specification φ^k. We observe that 3^k MILPs are generated for each k when using a compositional encoding; it becomes increasingly necessary to spread the computational load across the available worker processes especially when checking for infeasibility. The speed-up is most noticeable for $\dot{h}_0 = -19.5$.

We expect that Comp-Par is in general more performant when checking for feasibility. For Comp-Seq, we observed that the first MILP checked in the for-loop of Algorithm 2 was feasible, causing the loop to return early, giving quicker feasibility checks compared to Comp-Par. We observe that Monolithic is overall the least performant encoding, with several cases of timeouts when checking for infeasibility of the generated MILPs for $k \geq 4$. Although for VerticalCAS, the unsafe region was entered and eventually escaped, the performance of our compositional procedure exemplifies the tractability of finding shallow bugs in a faulty system.

We are unable to present a comparison with other tools because, as far as we are aware, no other tool supports branching models and CTL specifications as we do here. We use double-precision floating point numbers for representing real values. For the MILP solver that we use for our back-end, Gurobi, we use the default tolerance level of 10^{-6}, which represents the amount of numerical error allowed on a constraint while still considering it "satisfied". We rely on Gurobi for dealing with any further numerical issues. Note also that our encoding is more efficient than [2], which does not use symbolic linear relaxation for their neural network encoding nor interval arithmetic-based bounds propagation for MILP variables.

5 CONCLUSIONS

As we argued in Section 1, forthcoming autonomous systems will make greater use of machine learning methods; therefore there is an urgent need to develop techniques aimed at providing guarantees on the resulting behaviour of such systems. While the benefits of formal methods have long been recognised, and they have found large adoption in safety-critical systems as well as in industrial-scale software, there have been few efforts to introduce verification techniques for systems driven by neural networks.

In this paper we defined a system composed of a neural agent driven by deep feed-forward neural networks interacting with a non-deterministic environment. The resulting system displays branching evolutions. We defined and studied the resulting verification problem. While the problem is undecidable for full reachability, we isolated a fragment of the temporal language and showed that its corresponding verification problem is in coNExpTime. We developed and reported on a toolkit which includes a novel parallel algorithm to verify temporal properties of the complex environment defined in the VerticalCAS scenario. As demonstrated, while the parallel algorithm remains complete, it offers considerable advantages over its sequential counterpart when searching for counterexamples to bounded safety specifications in concrete examples.

In future work we plan to extend the framework to multiple agents operating in an environment.

ACKNOWLEDGMENTS

This work is partly funded by DARPA under the Assured Autonomy programme (FA8750-18-C-0095). Alessio Lomuscio is supported by a Royal Academy of Engineering Chair in Emerging Technologies.

REFERENCES

[1] M. E. Akintunde, A. Kevorchian, A. Lomuscio, and E. Pirovano. 2019. Verification of RNN-Based Neural Agent-Environment Systems. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI19)*. AAAI Press, 6006–6013.

[2] M. E. Akintunde, A. Lomuscio, L. Maganti, and E. Pirovano. 2018. Reachability Analysis for Neural Agent-Environment Systems. In *Proceedings of the 16th International Conference on Principles of Knowledge Representation and Reasoning (KR18)*. AAAI Press, 184–193.

[3] A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu. 2003. Bounded Model Checking. *Advances in Computers* 58 (2003), 117–148.

[4] R. H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. 2006. Verifying Multi-agent Programs by Model Checking. *Autonomous Agents and Multi-Agent Systems* 12, 2 (2006), 239–256.

[5] R. .R. Bunel, I. Turkaslan, P. Torr, P. Kohli, and P. K. Mudigonda. 2018. A Unified View of Piecewise Linear Neural Network Verification. In *Proceedings of the 31st Annual Conference on Neural Information Processing Systems (NeurIPS18)*. Curran Associates, Inc., 4790–4799.

[6] E. Clarke, A. Biere, R. Raimi, and Y. Zhu. 2001. Bounded Model Checking Using Satisfiability Solving. *Formal Methods in System Design* 19, 1 (2001), 7–34.

[7] E. M. Clarke, O. Grumberg, and D. A. Peled. 1999. *Model Checking*. The MIT Press, Cambridge, Massachusetts.

[8] C. D'Ambrosio, A. Lodi, and S. Martello. 2010. Piecewise linear approximation of functions of two variables in MILP models. *Operations Research Letters* 38, 1 (2010), 39–46.

[9] T. T. Doan, Y. Yao, N. Alechina, and B. Logan. 2014. Verifying heterogeneous multi-agent programs. In *Proceedings of the 13th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS14)*. IFAAMAS, 149–156.

[10] S. Dutta, X. Chen, and S. Sankaranarayanan. 2019. 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 (HSCC19)*. ACM, 157–168.

[11] R. Ehlers. 2017. Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks. In *Proceedings of the 15th International Symposium on Automated Technology for Verification and Analysis (ATVA17) (Lecture Notes in Computer Science)*, Vol. 10482. Springer, 269–286.

[12] E. A. Emerson, A. K. Mok, A. P. Sistla, and J. Srinivasan. 1992. Quantitative temporal reasoning. *Real-Time Systems* 4, 4 (1992), 331–352.

[13] P. Gammie and R. van der Meyden. 2004. MCK: Model Checking the Logic of Knowledge. In *Proceedings of 16th International Conference on Computer Aided Verification (CAV04) (Lecture Notes in Computer Science)*, Vol. 3114. Springer, 479–483.

[14] I. Griva, S. Nash, and A. Sofer. 2009. *Linear and nonlinear optimization*. Vol. 108. Siam.

[15] Z. Gu, E. Rothberg, and R. Bixby. 2016. Gurobi Optimizer Reference Manual. http://www.gurobi.com. (2016).

[16] S. S. Haykin. 1999. *Neural Networks: A Comprehensive Foundation*. Prentice Hall.

[17] C. Huang, J. Fan, W. Li, X. Chen, and Q. Zhu. 2019. ReachNN: Reachability Analysis of Neural-Network Controlled Systems. *ACM Transactions on Embedded Computing Systems (TECS)* 18, 106 (2019), 1–22.

[18] K.J. Hunt, D. Sbarbaro, R. Zbikowski, and P.J. Gawthrop. 1992. Neural networks for control systems–A survey. *Automatica* 28, 6 (1992), 1083–1112.

[19] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee. 2019. 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 (HSCC19)*. 169–178.

[20] K. Julian, J. Lopez, J. Brush, M. Owen, and M. Kochenderfer. 2016. Policy Compression for Aircraft Collision Avoidance Systems. In *Proceedings of the 35th Digital Avionics Systems Conference (DASC16)*. IEEE, 1–10.

[21] K. D. Julian and M. J. Kochenderfer. 2019. A Reachability Method for Verifying Dynamical Systems with Deep Neural Network Controllers. *CoRR* abs/1903.00520 (2019).

[22] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Proceedings of the 29th International Conference on Computer Aided Verification (CAV17) (Lecture Notes in Computer Science)*, Vol. 10426. Springer, 97–117.

[23] P. Kouvaros and A. Lomuscio. 2016. Parameterised Verification for Multi-Agent Systems. *Artificial Intelligence* 234 (2016), 152–189.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the 26th Conference on Neural Information Processing Systems (NIPS12)*. Curran Associates, Inc., 1097–1105.

[25] A. Lomuscio and L. Maganti. 2017. An approach to reachability analysis for feed-forward ReLU neural networks. *CoRR* abs/1706.07351 (2017).

[26] A. Lomuscio, H. Qu, and F. Raimondi. 2017. MCMAS: A Model Checker for the Verification of Multi-Agent Systems. *Software Tools for Technology Transfer* 19, 1 (2017), 9–30.

[27] P. Maes. 1993. Modeling adaptive autonomous agents. *Artificial life* 1, 1–2 (1993), 135–162.

[28] V. Nair and G. E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML10)*. Omnipress, 807–814.

[29] NANESVerify. 2020. Neural Agent operating on a Non-deterministic Environment System Verify, https://vas.doc.ic.ac.uk/software/neural/. (2020).

[30] N. Narodytska. 2018. Formal Analysis of Deep Binarized Neural Networks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence, (IJCAI18)*. IJCAI, 5692–5696.

[31] W. Penczek and A. Lomuscio. 2003. Verifying Epistemic Properties of multi-agent systems via bounded model checking. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multi-agent systems (AAMAS03)*. IFAAMAS, 209–216.

[32] W. Penczek, B. Woźna, and A. Zbrzezny. 2002. Bounded Model Checking for the Universal Fragment of CTL. *Fundamenta Informaticae* 51, 1-2 (2002), 135–156.

[33] J. Redmon, S. K. Divvala, R. B. Girshick, and Ali. Farhadi. 2016. You Only Look Once: Unified, Real-Time Object Detection. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR16)* (2016), 779–788.

[34] R. S. Sutton and A. G. Barto. 1998. *Reinforcement Learning – An Introduction*. MIT Press.

[35] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. 2014. Intriguing properties of neural networks. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR14)*.

[36] V. Tjeng, K. Xiao, and R. Tedrake. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *Proceedings of the 7th International Conference on Learning Representations (ICLR19)*.

[37] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana. 2018. Efficient Formal Safety Analysis of Neural Networks. In *Proceedings of the 32nd Conference on Neural Information Processing Systems (NIPS18)*. Curran Associates, Inc., 6367–6377.

[38] W. Winston. 1987. *Operations research: applications and algorithms*. Duxbury Press.

[39] W. Xiang, H. Tran, J. A. Rosenfeld, and T. T. Johnson. 2018. Reachable Set Estimation and Safety Verification for Piecewise Linear Systems with Neural Network Controllers. In *2018 Annual American Control Conference (ACC)*. AACC, 1574–1579.