Proceedings of Machine Learning Research 288:1–15, 2025 2nd International Conference on Neuro-symbolic Systems (NeuS)

# Neuro-Symbolic Behavior Trees (NSBTs) and Their Verification

**Serena S. Serbinowska** SERENA.SERBINOWSKA@VANDERBILT.EDU
**Diego Manzanas Lopez** DIEGO.MANZANAS.LOPEZ@VANDERBILT.EDU
**Dung Thuy Nguyen** DUNG.T.NGUYEN@VANDERBILT.EDU
**Taylor T. Johnson** TAYLOR.JOHNSON@VANDERBILT.EDU
*Vanderbilt University, Nashville TN 37235, USA*

**Editors:** G. Pappas, P. Ravikumar, S. A. Seshia

## Abstract

Neural networks have proven to be incredibly powerful and useful in a variety of domains, but are also often opaque and difficult to reason about. This is undesirable in safety-critical systems. An approach to help mitigate this is to utilize a neuro-symbolic approach that combines the power of neural networks and symbolic structures. In this paper, we present Neuro-Symbolic Behavior Trees (NSBTs). NSBTs are behavior trees that utilize neural networks. We provide several examples of NSBTs, including grid-world examples and a representation of a portion of ACAS Xu, an aircraft collision avoidance system. The grid world example considers over 6 million input states for the neural network, while the ACAS Xu example features 5 networks, each with 6 layers of 50 neurons. Additionally, we implemented support for NSBTs in our BehaVerify software tool, and verify certain safety and liveness properties for these NSBTs. Our verification approach also demonstrates how future improvements could be made using existing neural network verification techniques.

**Keywords:** Formal Model, Neural Networks, Behavior Trees, Verification,

## 1. Introduction

Behavior trees ($BTs$) are high level controllers that have become increasingly popular in robotics. Hallen et al. (2024) presents lessons learned from using $BTs$ in a robotic system that assembles and places explosive charges while Rocamora et al. (2024) describes controlling drones that inspect structures. Wu et al. (2024) uses $BTs$ for sensitive machine insertion tasks. Surveys such as Shin and Jung (2024) and Iovino et al. (2022) provide further examples.

Given the power of machine learning, it is natural to wonder how it can be used in conjunction with $BTs$. Several papers present strategies for using large language models to generate $BTs$ Li et al. (2024). Others propose methods for generating $BTs$ through reinforcement learning. We are taking a different approach. We are interested in $BTs$ that use neural networks ($NNs$). To that end, we formally define Neuro-Symbolic Behavior Trees ($NSBTs$) as a subclass of a $BTs$ known as Stateful Behavior Trees ($SBT$) Serbinowska et al. (2024b). $NSBTs$ can call $NNs$ and use the output to determine what action should be taken or to augment the value of a variable.

**Neuro-symbolic Artificial Intelligence and Systems** Garcez and Lamb (2023) highlighted the need for trustworthiness, interpretability, and accountability in AI systems; neuro-symbolic approaches can help address this. Neuro-symbolic AI, which integrates $NNs$ with symbolic reasoning, has seen increased adoption due to its ability to combine the strengths of both approaches Garcez and Lamb (2023); Sheth et al. (2023); Barnes and Hutson (2024). Neuro-symbolic systems have been utilized to improve diagnostic accuracy and personalize treatment plans Barnes and Hutson (2024) and have improved stability and safety in complex driving scenarios Sun et al. (2021); Gomaa and Feld (2023). More recently, a neuro-symbolic system has been successfully applied in the realm of visual question answering and natural language processing Mao et al. (2019); Hamilton et al. (2022).

**Neural Network Verification** $NN$ verification Tran et al. (2019); Johnson et al. (2024); Lopez et al. (2023, 2024); Katz et al. (2017, 2019) aims to ensure the correctness, robustness, and reliability of neural models.Various verification techniques have been used to verify properties like adversarial robustness, stability, and safety. NNV Tran et al. (2019); Johnson et al. (2024); Lopez et al. (2023, 2024) employs reachability analysis to verify safety and robustness for feedforward and convolutional networks while Reluplex Katz et al. (2017) and Marabou Katz et al. (2019) extend the simplex algorithm to handle piece-wise linear constraints introduced by ReLU activation functions, enabling effective verification of safety conditions. Recent advancements, such as branch-and-bound approaches Wang et al. (2021); Shi et al. (2025), further enhance verification scalability and effectiveness for nonlinear activation functions. A critical challenge in $NN$ verification lies in handling numerical precision, as floating-point errors can lead to unsound verification results, which are exploitable in practice Daggitt et al. (2024). Recent works address this by exploring verification under floating-point arithmetic, explicitly accounting for rounding errors and numerical stability Henzinger et al. (2021), or by focusing on fixed-point representations, which are crucial for embedded systems due to their deterministic behavior and efficiency Jia and Rinard (2020, 2021).

**Behavior Tree Verification** Various methods (model checking, runtime monitoring, and others) have been introduced to ensure the correctness, safety, and reliability of $BT$s in dynamic and complex environments. Biggar and Zamani (2020) introduced a formal verification framework based on Linear Temporal Logic (LTL), encoding $BT$s and their properties as logical formulae and reducing the verification problem to LTL satisfiability. ArcadeBT Henn et al. (2022) automates the verification process by encoding $BT$s as linearly constrained horn clauses and using the Z3 solver de Moura and Bjørner (2008) to verify safety properties. Colledanchise et al. (2021) formalizes $BT$s using program graphs and applies runtime monitoring to ensure correct behavior of a $BT$. Serbinowska et al. (2024a) developed a methodology for generating flexible runtime monitors that handle LTL specifications and integrate with BehaVerify Serbinowska and Johnson (2022) for formal verification. Wang et al. (2024) introduced a novel approach using the Behavior-Interaction-Priority framework to model $BT$s and verify formal properties. Existing methods often struggle with scalability, expressiveness, or applicability to real-world systems. Furthermore, there is a lack of integrated tools that seamlessly combine $BT$ design, execution, and verification. These limitations motivate our research, which aims to address these gaps by proposing a novel framework for $BT$ verification that improves scalability, expressiveness, and usability.

**Learning Behavior Trees** Another ML is to learn the $BT$ via a method like reinforcement-learning as is done in Banerjee (2018). Our work fundamentally differs in that we are interested in verifying a $BT$ that makes use of a $NN$ while these approaches focus on learning a $BT$.

**Neuro-Symbolic Behavior Trees** $NSBT$s are not new; indeed Sprague and Ögren (2022) introduces a strategy for combining an existing ML controller with an existing $BT$. While this work is related, the details are different; it creates a new $BT$ from the existing controllers that prioritized the ML controller, but utilizes safeguards to swap to the traditional controllers if needed. We verify properties about $BT$s that utilize $NN$s, regardless of how they are structured, created, or learned.

**Contributions** We provide a formal definition of $NSBT$s and note that they are a specific case of $SBT$s. We provide various examples of $NSBT$s, including a grid world example and a simplified version of ACAS Xu (an aircraft collision avoidance system) Julian et al. (2016). For grid world, 6250000 distinct inputs to a $NN$ were considered. The ACAS Xu example features 5 $NN$s, each with 6 hidden layers of 50 neurons each. We implement $NSBT$s in the Domain Specific Language (DSL) of BehaVerify Serbinowska et al. (2024b). We then used BehaVerify and nuXmv Cavada et al. (2014) to verify safety and liveness properties for $NSBT$s. Our tool can be found at [1].

## 2. Preliminaries

**Neural Networks** $NN$s are computational models that are widely used for tasks such as classification, regression, and function approximation due to their ability to learn complex patterns from data. A fundamental type of $NN$ is the Feed-Forward Neural Network (FNN), where data flows in one direction, from the input layer to the output layer through multiple hidden layers. In FNN, each neuron in the layer $k-1$ is connected to neurons in the next layer $k$ via weights $W_{k,k-1}$ and biases $b_k$. The output is passed through an activation function $f$ at each layer. Mathematically, the output of a neuron $i$ is defined by: $y_i = f(\Sigma_{j=1}^{n} \omega_{ij} x_j + b_i)$ where $x_j$ is the $j^{th}$ input of the $i^{th}$ neuron, $\omega_{ij}$ is the weight from the $j^{th}$ input to the $i^{th}$ neuron, $b_i$ is the bias of the $i^{th}$ neuron. In this paper, our activation function $f$ will be ReLU, defined as $\mathrm{ReLU}(x) = \max(0, x)$.

**Behavior Trees** A Behavior Tree ($BT$) is a rooted tree. It does nothing until an external signal called a 'tick' arrives. When a tick arrives, the root becomes 'active'. The tick ends when the root returns a status. At any time during the tick, exactly one node is active. When a node is active, it will either cause one of its children to become active or return a status to its parent. The possible statuses are success ($S$), failure ($F$), and running ($R$). Until a node finishes, it is invalid ($I$). Note that running ($R$) means the node has finished! For example, suppose a node is subscribed to a topic; it returns $S$ if it receives 'all clear', $F$ if it receives 'error', and $R$ if a message has not arrived. In each case, the node finished its task; $R$ is used to signify that the tree cannot assume $S$ **or** $F$. Refer to Figure 1 for an example $BT$ and execution as well as an explanation of how to read our $BT$ diagrams.

**Composite Nodes** Composite nodes (nodes with children) control the 'flow' through the $BT$. Execution follows depth-first traversal but composite nodes can cause portions of the tree to be skipped. There are three types of composite nodes: selector, sequence, and parallel. Parts of the tree are skipped when a child of a selector or sequence returns $R$, the child of a selector returns $S$, or the child of a sequence returns $F$. In these cases, the composite node will return with the status that caused the skip without running the remainder of its children. Parallel nodes never skip any of their children. Examples of this can be seen in Figure 1, where $Seq$ skips $NewGoal$ when $NeedGoal$ returns $F$. If no skips occur, a composite node will run children in order until it runs out of children, at which point it will return a status. For a selector, if no skips occur, it will return $F$. For a sequence, if no skips occur, it will return $S$. A parallel node uses a policy: for instance, with the 'success on all' policy a parallel node returns $S$ if and only if each of its children returns $S$.

**Leaf Nodes** Leaf nodes do not have children. Unlike composite nodes, users often define their own leaf nodes. Leaf nodes can change the values of blackboard variables and can return statues based on blackboard and environment variables (these are described below).

**Variables** We consider two categories of variables: blackboard and environment. The blackboard refers to the shared memory of the $BT$. Each node in the tree can access the blackboard. Blackboard variables do not change unless an action node changes them. We write blackboard variables in this color. The environment refers to everything outside of the $BT$. This could be the wind speed,

1. https://github.com/verivital/behaverify

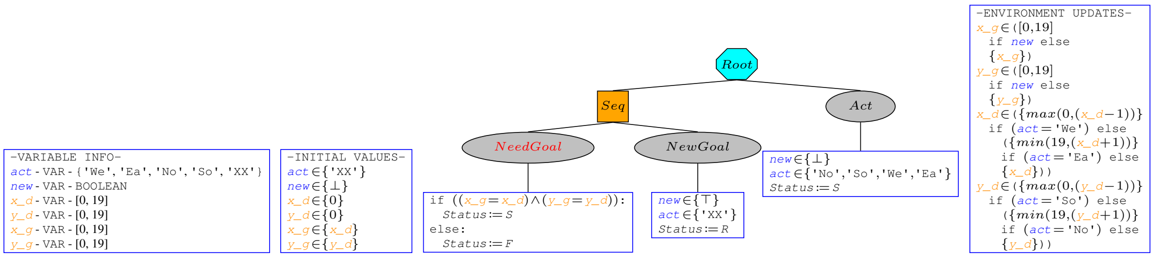


Figure 1: An example $BT$ that moves a drone in a random direction on a grid. If a goal is reached, a new one is generated. Each node is represented using a different shape and color: selectors are octagonal, sequences are rectangles, parallel nodes are parallelograms, and decorators are trapezoids. There are no parallel nodes or decorators in this example. Leaf nodes are ovals; checks have a red label and actions have a black label. A box of 'code' is beneath each leaf; it is executed from top to bottom. $\in$ is used to denote assignment to a value within a set; this allows for nondeterminism. The 'variable info' box contains information about variables; VARs can be updated, FROZENVARs keep an initial value, DEFINE are functions (e.g., if a DEFINE is equal to $1+v$ and the value of $v$ changes, then the value of the DEFINE will change), and NEURAL refers to neural networks. It also states the domain. The 'initial values' and 'environment updates' boxes are 'code' boxes. Environment updates take place between ticks; environment variables do not change during a tick.

| **Tick** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **t** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| **Active** | Root | Seq | Need | Seq | New | Seq | Root | Env | Root | Seq | New | Seq | Root | Act | Root | Env |
| **Ret** | $I$ | $I$ | $S$ | $I$ | $R$ | $R$ | $R$ | $I$ | $I$ | $I$ | $F$ | $F$ | $I$ | $S$ | $S$ | $I$ |
| $act$ | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | XX | No | No | No |
| $new$ | $\bot$ | $\bot$ | $\bot$ | $\bot$ | $\top$ | $\top$ | $\top$ | $\top$ | $\top$ | $\top$ | $\top$ | $\top$ | $\top$ | $\bot$ | $\bot$ | $\bot$ |
| $x\_d$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $y\_d$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 |
| $x\_g$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| $y\_g$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |

Table 1: A table illustrating a hypothetical execution of the $BT$ shown in Figure 1. **Tick** refers to the number of times the tree has been 'called'. **t** refers to the number of timesteps taken. At each timestep, a single node is active; this is what **Active** refers to. We use shorthand for some nodes and Env refers to the environment update, which occurs between ticks. Nodes return one of $S$, $F$, or $R$ when they finish ($I$ means the node has not yet finished); this is tracked by **Ret**.

the temperature outside, or a data request from a connection client. Crucially, we assume that environment variables only change between ticks. We write environment variables in this color.

**Formal Definition** In Serbinowska et al. (2024b), we presented a formal definition for Stateful Behavior Trees ($SBT$s). Here we will provide a simplified definition of $SBT$s and then explain how $NSBT$s relate to them. A $SBT$ is a tuple $(V,r,E,S_{BT},s_{BT},\Sigma_{BT},\delta_{BT})$ such that:

- $(V,r,E)$ is a rooted tree. Here $V$ is the set of nodes (vertices) in the tree and $r$ is the root node. $E$ is a function that maps nodes to ordered sequences of nodes, representing the children of nodes. E.G., $E(A)=[B,C,D]$ means $A$'s children are $B$, $C$, and $D$ in that order.
- $S_{BT}$ is a set representing the possible states of the blackboard of $SBT$. For instance, if we had two variables, one a Boolean and one an integer between 1 and 3, this set would be $\{(\top,1),(\top,2),(\top,3),(\bot,1),(\bot,2),(\bot,3)\}$. $s_{BT}\in S_{BT}$ is the initial state of the blackboard.

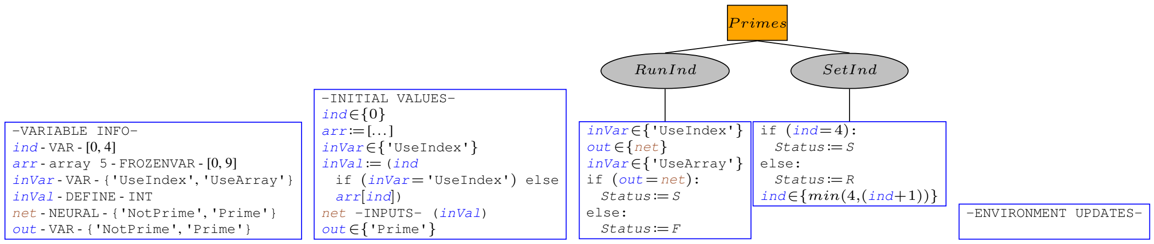


Figure 2: A basic $NSBT$ that makes use of the $NN$ $net$. $net$ takes as input a single integer ($inVar$) and outputs 'prime' or 'not prime'. Within the tree, we simply use $net$ to denote calling the network with the appropriate input. The $NSBT$ makes use of this to determine if an array of numbers ($arr$) obeys the property that $\forall i \in \mathbb{Z}, 0 \leq i < len(arr) \implies (prime(i) \iff prime(arr[i]))$.

- $\Sigma_{BT}$ is a set representing the possible inputs (the environment).
- $ST$ is the set of all functions $st : V \mapsto \{S, R, F, I\}$. Each $st \in ST$ is a function that maps each vertex to a status. $ST$ is not an element of the tuple; it arises from the elements.
- $\delta_{BT} : V \times ST \times S_{BT} \times \Sigma_{BT} \mapsto 2^{V \times ST \times S_{BT}}$. Here $2^{V \times ST \times S_{BT}}$ is the power set of $V \times ST \times S_{BT}$. The function maps to sets to allow for nondeterminism. This function takes as input the active node, a function representing the status of each node, the state of the blackboard, and external input from the environment and produces an active node, a function representing the current status of each node, and a state for the blackboard. This function obeys additional rules to ensure it represents how $BT$s work (e.g. the next active node is either the parent or a child of the current node).

A Neuro-Symbolic Behavior Tree ($NSBT$) is a $BT$ that utilizes at least one $NN$; that is to say a leaf node can use a $NN$ either to determine the status that will be returned ($S$, $F$, or $R$) or to determine the value of a variable in the blackboard. See Section 3 for examples. The definition for $SBT$s permits this behavior; $\delta_{BT}$ can depend on a $NN$ to determine either the status of the active node or the state of the blackboard. Thus $NSBT$s are a subset of $SBT$s.

While the existing definition of $SBT$s encompasses $NSBT$s, it is important to note that it is a broad and abstract definition. In particular, Serbinowska et al. (2024b) demonstrated that if the blackboard can store true mathematical integers, then $SBT$s are equivalent to Turing Machines. As such, our practical implementation of $SBT$s within BehaVerify utilizes a $DSL$ that greatly restricts what can be used within $SBT$s. We have expanded our $DSL$ to allow for $NN$s to be used in BehaVerify. See Section 4 for a description of how $NN$s are handled in BehaVerify and a discussion of verification results for the example $NSBT$s.

## 3. Examples

We provide three examples of $NSBT$s: prime position, grid world, and ACAS Xu. Prime position is meant to help introduce and illustrate how $NSBT$s function. Grid world helps demonstrate some of the performance differences between our various approaches of encoding $NN$s. ACAS Xu illustrates how $NSBT$s can be used to handle real world tasks. We write networks in this color.

**Prime Position** The prime position example (see Figure 2) is a basic introductory example. This $NSBT$ features a very basic $NN$; it accepts as input a single integer and classifies it as prime or not prime. It has been trained on integers between 0 and 9. The $NSBT$ checks if an array of numbers obeys the property that the $i^{th}$ number in the array is prime if and only if $i$ is prime.

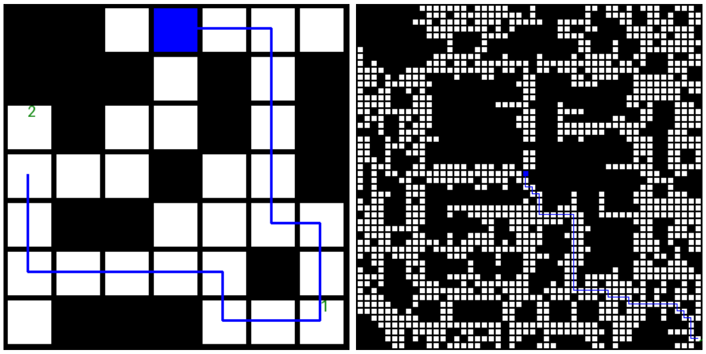


Figure 3: A drone (blue) avoids obstacles (black) in order to reach a target (green numbers). When a target is reached, a new target is created. See Figure 4 for the $NSBT$ that controls the drone.

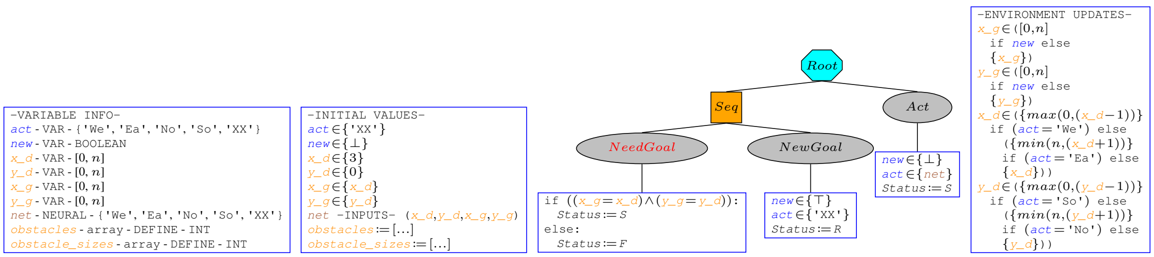


Figure 4: A $NSBT$ that moves a drone $(x_d,y_d)$ on a grid towards a target $(x_g,y_g)$. $n$ depends on the size of the grid. If the drone reaches the target it requests a new target. It uses the network $net$ to determine the direction the drone should go in based on the location of the drone and target. It was trained using A* to avoid obstacles and take an optimal path towards the target, though the training is grid specific. Note that the obstacles were determined before training the network.

**Grid World** In the grid world example, a drone operates on a 2d-grid. It moves one square at a time (up, down, left, right) towards a target while avoiding obstacles. When it reaches a target, a new target is generated. See Figure 3 for examples. The $NSBT$ that controls the drone can be seen in Figure 4.

**ACAS Xu** ACAS Xu is optimized for unmanned aircraft systems and issues turn rate advisories to remote pilots to avoid near midair collisions Marston and Baca (2015), defined as separation less than 100 ft vertically and 500 ft horizontally Holland et al. (2013). ACAS Xu assigns turn rate advisories based on a set of input variables as described in Table 2. The first five variables describe 2D considerations, the sixth variable brings the scenario into 3D (altitude difference), and the seventh variable promotes advisory selection consistency.

Developed in Julian et al. (2016) and evaluated in Katz et al. (2017), 45 separate $NN$s were used to compress the lookup table. Each network is denoted $N_{\gamma,\beta}$, where $\gamma$ corresponds to the index (1 to 5) of a specific value of previous advisory $a_{prev} \in \{C,WL,WR,SL,SR\}$ and $\beta$ corresponds to the index (1 to 9) of a specific value of time to loss of vertical separation $\tau \in \{0,1,5,10,20,40,60,80,100\}$ seconds. Thus, $N_{5,1}$ corresponds to a $NN$ in which $a_{prev} = SR$ and $\tau = 0$. Each network receives inputs for the remaining five state variables ($\rho$, $\theta$, $\psi$, $v_{own}$, and $v_{int}$) and outputs a value associated with each of the five output variables ($\{C,WL,WR,SL,SR\}$). These represent actions: $C$ means do nothing, $WL$ means 1.5 deg/s left, $WR$ means 1.5 deg/s right, $SL$ means 3 deg/s left, and $SR$ means 3 deg/s right. Each network has six hidden ReLU layers of 50 neurons Julian et al. (2016). Thus each network has five inputs, five outputs, and six hidden layers of 50 neurons.

| Variable | Units | Description |
|---|---|---|
| $\rho$ | ft | distance between ownship and intruder |
| $\theta$ | rad | angle to intruder w.r.t ownship heading |
| $\psi$ | rad | heading of intruder w.r.t ownship heading |
| $v_{own}$ | ft/s | velocity of ownship |
| $v_{int}$ | ft/s | velocity of intruder |
| $\tau$ | s | time until loss of vertical separation |
| $a_{prev}$ | deg/s | previous advisory |

Table 2: Input state variables in ACAS Xu. Note that $\tau$ and $a_{prev}$ are used only to determine which $NN$ is used. The remaining inputs are used as inputs to the $NN$s.

In this manuscript we model a simplified version of ACAS Xu as a $NSBT$ (see [2]). We assume that both aircraft are flying at the same fixed elevation, so only 5 $NN$s are considered, corresponding to $\tau$=0 ($N_{a_{prev},1}$). We created two models in BehaVerify: a simple model for 'local robustness' and a basic closed-loop model.

**Definition 1 (Local Robustness)** *Let $f:\mathbb{R}^n \to \mathbb{R}^m$ be a $NN$, and let $x \in \mathbb{R}^n$ be an input to the network. The network is locally robust at $x$ with respect to a perturbation radius $\epsilon > 0$ if for all $x' \in \mathbb{R}^n$ such that $\|x' - x\| \le \epsilon$, the output of the network remains unchanged. Mathematically, this can be expressed as: $\forall x' \in \mathbb{R}^n, \left\|x' - x\right\| \le \epsilon \Longrightarrow f\left(x'\right) = f(x)$, where $\|\cdot\|$ is a norm (e.g., $L_2$-norm or $L_\infty$-norm) defining the distance between inputs, and $\epsilon$ is the maximum allowable perturbation.*

Intuitively, a $NN$ is locally robust at a given input if every other input that is 'close' to that input produces the same output. Taking inspiration from this, we created a model where each input to the $NN$s is restricted to a small region of integers. Obviously this isn't the same as local robustness; we are limiting our inputs based on certain integer values. Details about the verification of this model can be found in Section 4.

**Closed-Loop Model** Unlike the 'local robustness' model, the closed-loop model seeks to 'simulate' how ACAS Xu would work in practice. That is to say, the closed-loop model has state variables representing the positions of the aircraft and updates them based on their headings and speeds. Additional details about the closed-loop model can be found in Subsection 4.

## 4. Verification and Results

First, we note that the experiments were ran on a machine with 32gb of RAM and a 13th Gen Intel(R) Core(TM) i7-13700K. The experiments and the code used to generate results can be found at [3]. The input to BehaVerify (Serbinowska et al. (2024b)) is written using a $DSL$; this input contains an $SBT$ and the environment it operates within. Specifications can be written using invariants, Linear Temporal Logic (LTL), and Computational Tree Logic (CTL). BehaVerify can translate the input into Python code or a nuXmv (a state-based model checker) model Cavada et al. (2014). To support $NSBT$s in BehaVerify, we had to represent $NN$s in nuXmv. We implemented three strategies to accomplish this: float, fixed, and table. The table strategy records and stores the output of the $NN$ for all inputs. The inputs and outputs are included in nuXmv as a lookup table, replacing the $NN$.

The fixed strategy involves simulating the $NN$ within nuXmv. Each weight and bias is stored using a fixed-point representation and the output of the network is then calculated directly. Suppose we want to multiply 1.5 and .32 using 6 digits total with 3 for the fractional part. We would store these values as 001500 and 000320 and multiply them to get 480000. 'Digit shifting' the result to the right by the number of digits used for fractional part yields 000480. This value represents .48, the

2. https://github.com/verivital/behaverify/blob/main/REPRODUCIBILITY/2025_NEUS/examples/AcasXu/acasxu_SETPOINT.pdf
3. https://github.com/verivital/behaverify/tree/main/REPRODUCIBILITY/2025_NEUS

| Neurons | 100-35 | 140-48 | Table | Neurons | 100-35 | 140-48 | Table | Neurons | 100-35 | 140-48 | Table |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 100 | 0.240 | 0.267 | 0.257 | 100 | 53.15 | 54.00 | 0.07 | 100 | 54.27 | 54.12 | 0.20 |
| 150 | 0.251 | 0.275 | 0.260 | 150 | 64.79 | 66.30 | 0.07 | 150 | 65.11 | 66.11 | 0.19 |
| 200 | 0.259 | 0.289 | 0.261 | 200 | 96.36 | 97.40 | 0.06 | 200 | 96.51 | 98.70 | 0.19 |
| 250 | 0.275 | 0.300 | 0.263 | 250 | 123.09 | 129.30 | 0.07 | 250 | 125.10 | 124.61 | 0.20 |
| 300 | 0.283 | 0.313 | 0.266 | 300 | 153.37 | 154.53 | 0.07 | 300 | 153.89 | 151.33 | 0.19 |

Table 3: Left: time to translate to .smv file. Center: time to verify the invariant. Right: time to verify the CTL. This is for the smaller grid (see Figure 3). Times are listed in seconds. A-B means fixed point with A bits in total and B for the fractional portion. For center and right, results include time to build the model in nuXmv and verify the specification; verification took about .01 seconds for the invariant and .15 seconds for CTL. The table approach is unaware of the size of the network (it only keeps track of inputs and outputs); this explains why it performs the same in all cases.

result of $1.5*.32$. In practice we use bits, not digits. Had we used 4 digits for the fractional part, then the result would have been $015000*003200 = 48000000$, resulting in overflow. The user configures the number of bits and it is the user's responsibility to make sure overflow does not occur. A floating point representation would help mitigate the overflow issue, but it proved too inefficient, most likely as a result of more complex multiplication logic, so we omit it here for brevity.

INVAR: $(x_d, y_d) \notin Obs$ CTL: $AG(((x_t, y_t) \in Obs) \vee (AF(x_d = x_t \wedge y_d = y_t)))$

**Grid World** For the $NSBT$ in Figure 4 we had two specifications (see above). $Obs$ refers to the set of obstacles, $AG$ stands for always globally, and $AF$ stands for always finally. The invariant states the drone is never in an obstacle. The CTL states it is always the case that either the target is inside an obstacle or is eventually reached. We ensured the drone does not start in an obstacle and that there are no 'unreachable' areas walled off by obstacles.

We start with the smaller grid (see Figure 3). BehaVerify first translates the input files written using the $DSL$ into .smv files for use with nuXmv. The .smv files are then used with nuXmv for verification. The timing results for this can be found in Table 3. Note that the fixed point method gets slower as the size of the network increases. Surprisingly, the results of Fixed-100-35 vs Fixed-140-48 are very similar. In the first case, we are storing each fixed point number using 100 bits, 35 of which are dedicated to the fractional portion. In the second case, its 140 and 48. This has a noticeable increase on file size (551.5 vs 666.8 KiB at 300 neurons), but the impact on performance is minuscule. Thus it is better to err on the side of caution and use more bits than is strictly necessary.Finally, we note that the table method boasts not only the best performance of the three methods, but is also resilient to large network sizes. This also presents an avenue for future work (see Section 5).

We also used the table method on the larger grid (see Figure 3). The invariant specification was verified in 29.32 seconds, 11.75 of which were spent building the model. After an hour, we terminated the CTL verification attempt. While the invariant and CTL verification times were comparable on the smaller grid, it is clear that the CTL specification is much more difficult to verify on larger grids. We note that there are 6250000 possible combinations of drone and target on the larger grid and we verified that the drone will never crash into an obstacle in under 30 seconds.

**Counterexamples** So far, our examples have had perfect networks. Imperfect networks can introduce errors. Consider the network visualization presented in Figure 5. Having imperfect networks will result in nuXmv finding counterexamples to our specifications, as seen in Figure 6.

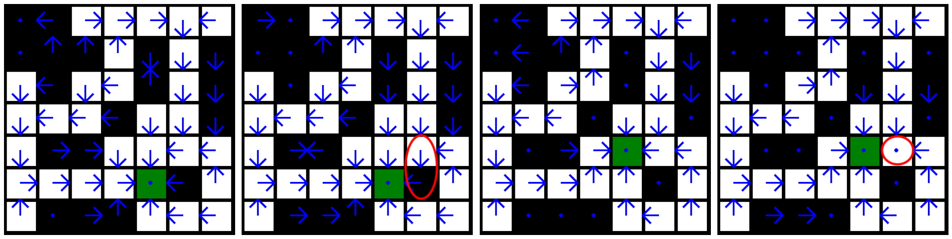

Figure 5: A visualization of two grid-world networks. They take as input the a 4-tuple representing the location of the drone and target. Green means target, black means obstacle. An arrow represents the direction the $NN$ would move the drone if it were in the square. A dot means no movement. The images with red ovals correspond to networks that make mistakes. In the second image the network can cause a collision (see red oval). In the last image the network can cause the drone to 'get stuck' (see red oval). The training data did not include scenarios where the drone was in an obstacle. nuXmv counter-example traces can be seen in Figure 6.

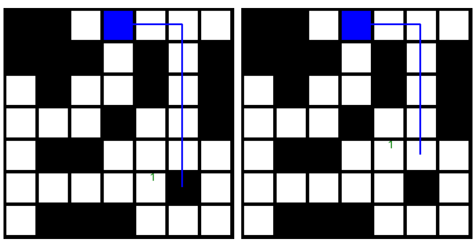

Figure 6: Counter-example traces generated by nuXmv for the incorrect networks in Figure 5. The blue square is the drone's starting locations. The blue line traces the path the drone took. The black squares represent obstacles. The green number represents the drone's target. Left: the drone crashes into an obstacle. Right: the drone never reaches the target, instead getting stuck.

**ACAS Xu** We only used the table method for ACAS Xu. We needed to normalize our inputs for ACAS Xu. For example, suppose the aircrafts are 50000 ft apart. Then, our actual input is $\frac{50000-19791.091}{60621} = 0.498...$. Since we were using the table method, this normalization was handled during the translation. Each of the 5 inputs to ACAS Xu was normalized in this manner.

**'Local Robustness'** For this model, we considered the following invariant conditions:

1. $(a_{prev} = C) \implies (a_{next} = WL)$
2. $(a_{prev} = SL) \implies (a_{next} = WL)$
3. $(a_{prev} = WL) \implies (a_{next} = WL)$
4. $(a_{prev} = WR) \implies (a_{next} = WR)$
5. $(a_{prev} = SR) \implies (a_{next} = WR)$
6. $(a_{prev} = C) \implies (a_{next} = SR)$

The first 5 are true and the last is false. In essence, they state that if the plane is turning right, then it should continue to slowly turn right. If it is turning left, then it should continue to slowly turn left. Note that the first 5 invariants are true only because we are considering a small region of space (a mimicry of local robustness). Here $a_{prev}$ refers to the value $a$ has at the start of the tick (the previous output of ACAS Xu) while $a_{next}$ refers to the value at the end of the tick (the current output of ACAS Xu). While we are using 'prev' and 'next' here, it is important to note that the actual encoding BehaVerify uses for a situation like this would not involve operators like next or previous; these truly are invariant specifications that have no temporal aspect within BehaVerify. For additional details, see Serbinowska et al. (2024b) for details about how the BehaVerify encoding works.

Table 4 is surprising; even though this model only has 456775 distinct $NN$ inputs, it performed far worse than the large grid world which has 6250000 distinct $NN$ inputs. We suspect this arises from the 5 $NN$s, each of which creates a table with 456775 entries. The fact that each of these networks also affects the same variable may create unexpected complexity in nuXmv.

| Ranges | Total | Translation | Build | Verification |
|---|---|---|---|---|
| [9975,100025],[-1,1],[89,91],[495,500],[700,705] | 16524 | 1.611 | 2.63 | 2.62 |
| [9950,100050],[-1,1],[89,91],[495,505],[695,705] | 109989 | 9.247 | 20.64 | 12.22 |
| [9925,100075],[-2,2],[88,92],[495,500],[700,705] | 456775 | 38.347 | 115.27 | — |

Table 4: This table shows timing results for ACAS Xu. Total is equal to the result of multiplying ranges, where ranges shows the ranges for $\rho$,$\theta$,$\psi$,$v_{own}$,$v_{int}$. Translation, build, and verification are all listed in seconds. Translation is the amount of time it takes to translate the input written using the $DSL$ into a .smv file for use with nuXmv. Build is the amount of time nuXmv takes to build the model. Verification is the amount of time nuXmv takes to verify the model. Note that verification for the largest model was aborted after 10 minutes.

**Closed Loop** We note that the closed-loop model for ACAS Xu is a proof of concept. The positions are heavily rounded, aircraft adjust heading instantaneously, and ACAS Xu is called every 6 seconds. Thus the invariant specification, $\rho \geq 200$, is checked every 6 seconds. It is possible that aircraft crash between those 6 seconds without our model noticing. In short, our closed-loop model of ACAS Xu cannot be used to argue for the correctness of ACAS Xu. It serves as a demonstration for how $NSBT$s can be used and provides groundwork for a verification approach that could be improved upon in the future. It took 2.15 seconds to translate the model to a .smv file, 40.40 seconds to build the model in nuXmv, and 8.88 seconds to verify the invariant specification. Note that closed loop verification was much harder than 'local robustness' and required aggressive simplification.

## 5. Conclusions and Future Work

We presented $NSBT$s, introduced several examples, and demonstrated that BehaVerify is capable of completing interesting verification tasks for the $NSBT$s using nuXmv. However, there is still work to be done. We would like to improve the performance of BehaVerify with respect to large networks. One approach is to utilize existing tools for $NN$ verification as nuXmv is not specialized for $NN$s. Instead of encoding the $NSBT$ with the $NN$s, we could use $NN$ verification on the $NN$s, and create an assume-guarantee compositional verification framework providing only the proven pre and post-conditions over the $NN$s for the encoding to nuXmv. This is not unlike the table approach; in the table approach, for a specific input, we record the output. This can be thought of as a guarantee; if this exact input is provided, the network will output this.

Additionally, our examples so far have focused on classification networks. This is because our models have been discrete in nature. nuXmv supports the use of reals; however, thus far our attempts to use reals with BehaVerify have yielded very poor performance results. As such, we are still exploring how to improve our support for regression networks (and reals in general).

**Acknowledgements** The material presented in this paper is based upon work supported by the National Science Foundation (NSF) through grant numbers 2220426 and 2220401, the Defense Advanced Research Projects Agency (DARPA) under contract number FA8750-23-C-0518, and the Air Force Office of Scientific Research (AFOSR) under contract numbers FA9550-22-1-0019 and FA9550-23-1-0135. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of AFOSR, DARPA, or NSF.

## References

Bikramjit Banerjee. Autonomous acquisition of behavior trees for robot control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, page 3460–3467. IEEE Press, 2018. doi: 10.1109/IROS.2018.8594083. URL https://doi.org/10.1109/IROS.2018.8594083.

Emily Barnes and James Hutson. Natural language processing and neurosymbolic ai: The role of neural networks with knowledge-guided symbolic approaches. *DS Journal of Artificial Intelligence and Robotics*, 2024. URL https://api.semanticscholar.org/CorpusID:268581074.

Oliver Biggar and Mohammad Zamani. A framework for formal verification of behavior trees with linear temporal logic. *IEEE Robotics and Automation Letters*, 5(2):2341–2348, 2020. doi: 10.1109/LRA.2020.2970634.

Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuxmv symbolic model checker. In *CAV*, pages 334–342, 2014.

Michele Colledanchise, Giuseppe Cicala, Daniele E. Domenichelli, Lorenzo Natale, and Armando Tacchella. Formalizing the execution context of behavior trees for runtime verification of deliberative policies. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9841–9848. IEEE Press, 2021. doi: 10.1109/IROS51168.2021.9636129.

Matthew L. Daggitt, Wen Kokke, Robert Atkey, Natalia Slusarz, Luca Arnaboldi, and Ekaterina Komendantskaya. Vehicle: Bridging the embedding gap in the verification of neuro-symbolic programs, 2024. URL https://arxiv.org/abs/2401.06379.

Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-78800-3. doi: 10.1007/978-3-540-78800-3_24.

Artur d'Avila Garcez and Luís C. Lamb. Neurosymbolic ai: the 3rd wave. *Artif. Intell. Rev.*, 56 (11):12387–12406, March 2023. ISSN 0269-2821. doi: 10.1007/s10462-023-10448-w. URL https://doi.org/10.1007/s10462-023-10448-w.

Amr Gomaa and Michael Feld. Towards adaptive user-centered neuro-symbolic learning for multimodal interaction with autonomous systems. In *Proceedings of the 25th International Conference on Multimodal Interaction*, ICMI '23, page 689–694, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400700552. doi: 10.1145/3577190.3616121. URL https://doi.org/10.1145/3577190.3616121.

Mattias Hallen, Matteo Iovino, Shiva Sander-Tavallaey, and Christian Smith. Behavior trees in industrial applications: A case study in underground explosive charging. In *2024 IEEE 20th International Conference on Automation Science and Engineering (CASE)*, pages 156–162, 2024. doi: 10.1109/CASE59546.2024.10711822.

Kyle Hamilton, Aparna Nayak, Bojan Bozic, and Luca Longo. Is neuro-symbolic ai meeting its promises in natural language processing? a structured review. *Semantic Web*, 15, 09 2022. doi: 10.3233/SW-223228.

Thomas Henn, Marcus Völker, Stefan Kowalewski, Minh Trinh, Oliver Petrovic, and Christian Brecher. Verification of behavior trees using linear constrained horn clauses. In Jan Friso Groote and Marieke Huisman, editors, *Formal Methods for Industrial Critical Systems*, pages 211–225, Cham, 2022. Springer International Publishing. ISBN 978-3-031-15008-1. doi: 10.1007/978-3-031-15008-1_14.

Thomas A. Henzinger, Mathias Lechner, and Đorđe Žikelić. Scalable verification of quantized neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(5):3787–3795, May 2021. doi: 10.1609/aaai.v35i5.16496. URL https://ojs.aaai.org/index.php/AAAI/article/view/16496.

Jessica E. Holland, Mykel J. Kochenderfer, and Wesley A. Olson. Optimizing the next generation collision avoidance system for safe, suitable, and acceptable operational performance. *Air Traffic Control Quarterly*, 21(3):275–297, 2013. doi: 10.2514/atcq.21.3.275. URL https://doi.org/10.2514/atcq.21.3.275.

Matteo Iovino, Edvards Scukins, Jonathan Styrud, Petter Ögren, and Christian Smith. A survey of behavior trees in robotics and ai. *Robotics and Autonomous Systems*, 154:104096, 2022. ISSN 0921-8890. doi: 10.1016/j.robot.2022.104096.

Kai Jia and Martin Rinard. Efficient exact verification of binarized neural networks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Kai Jia and Martin Rinard. Exploiting verified neural networks via floating point numerical error. In *Static Analysis: 28th International Symposium, SAS 2021, Chicago, IL, USA, October 17–19, 2021, Proceedings*, page 191–205, Berlin, Heidelberg, 2021. Springer-Verlag. ISBN 978-3-030-88805-3. doi: 10.1007/978-3-030-88806-0_9. URL https://doi.org/10.1007/978-3-030-88806-0_9.

Taylor T. Johnson, Diego Manzanas Lopez, and Hoang-Dung Tran. Tutorial: Safe, secure, and trustworthy artificial intelligence (ai) via formal verification of neural networks and autonomous cyber-physical systems (cps) with nnv. In *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S)*, pages 65–66, 2024. doi: 10.1109/DSN-S60304.2024.00027.

Kyle D. Julian, Jessica Lopez, Jeffrey S. Brush, Michael P. Owen, and Mykel J. Kochenderfer. Policy compression for aircraft collision avoidance systems. *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pages 1–10, 2016. URL https://api.semanticscholar.org/CorpusID:3123038.

Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In Rupak Majumdar and Viktor Kunčak, editors, *Computer Aided Verification*, pages 97–117, Cham, 2017. Springer International Publishing. ISBN 978-3-319-63387-9.

Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, David L. Dill, Mykel J. Kochenderfer, and Clark Barrett. The marabou framework for verification and analysis of deep neural networks. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, pages 443–452, Cham, 2019. Springer International Publishing. ISBN 978-3-030-25540-4.

Fu Li, Xueying Wang, Bin Li, Yunlong Wu, Yanzhen Wang, and Xiaodong Yi. A study on training and developing large language models for behavior tree generation, 2024. URL https://arxiv.org/abs/2401.08089.

Diego Manzanas Lopez, Sung Woo Choi, Hoang-Dung Tran, and Taylor T. Johnson. Nnv 2.0: The neural network verification tool. In Constantin Enea and Akash Lal, editors, *Computer Aided Verification*, pages 397–412, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-37703-7.

Diego Manzanas Lopez, Matthias Althoff, Luis Benet, Clemens Blab, Marcelo Forets, Yuhao Jia, Taylor T Johnson, Manuel Kranzl, Tobias Ladner, Lukas Linauer, Philipp Neubauer, Sophie Neubauer, Christian Schilling, Huan Zhang, and Xiangru Zhong. Arch-comp24 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants. In Goran Frehse and Matthias Althoff, editors, *Proceedings of the 11th Int. Workshop on Applied Verification for Continuous and Hybrid Systems*, volume 103 of *EPiC Series in Computing*, pages 64–121. EasyChair, 2024. doi: 10.29007/mxld. URL /publications/paper/WsgX.

Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences From Natural Supervision. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJgMlhRctm.

Mike Marston and Gabe Baca. Acas-xu initial self-separation flight tests. Technical report, NASA Armstrong Flight Research Center, 2015.

Bernardo Martinez Rocamora, Paulo V. G. Simplicio, and Guilherme A. S. Pereira. A behavior tree approach for battery-aware inspection of large structures using drones. In *2024 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 234–240, 2024. doi: 10.1109/ICUAS60882.2024.10557083.

Serena S. Serbinowska and Taylor T. Johnson. Behaverify: Verifying temporal logic specifications for behavior trees. In *Software Engineering and Formal Methods: 20th International Conference, SEFM 2022, Berlin, Germany, September 26-30, 2022, Proceedings*, pages 307–323, Berlin, Heidelberg, 2022. Springer-Verlag. ISBN 978-3-031-17107-9. doi: 10.1007/978-3-031-17108-6_19.

Serena S. Serbinowska, Nicholas Potteiger, Anne M. Tumlin, and Taylor T. Johnson. Verification of behavior trees with contingency monitors. In Matt Luckcuck and Mengwei Xu, editors, Proceedings Sixth International Workshop on *Formal Methods for Autonomous Systems,* Manchester, UK, 11th and 12th of November 2024, volume 411 of *Electronic Proceedings in Theoretical Computer Science*, pages 56–72. Open Publishing Association, 2024a. doi: 10.4204/EPTCS.411.4.

Serena S. Serbinowska, Preston Robinette, Gabor Karsai, and Taylor T. Johnson. Formalizing stateful behavior trees. In Matt Luckcuck and Mengwei Xu, editors, Proceedings Sixth International Workshop on *Formal Methods for Autonomous Systems,* Manchester, UK, 11th and 12th of November 2024, volume 411 of *Electronic Proceedings in Theoretical Computer Science*, pages 201–218. Open Publishing Association, 2024b. doi: 10.4204/EPTCS.411.14.

Amit Sheth, Kaushik Roy, and Manas Gaur. Neurosymbolic artificial intelligence (why, what, and how). *IEEE Intelligent Systems*, 38(3):56–62, 2023. doi: 10.1109/MIS.2023.3268724.

Zhouxing Shi, Qirui Jin, Zico Kolter, Suman Jana, Cho-Jui Hsieh, and Huan Zhang. Neural network verification with branch-and-bound for general nonlinearities, 2025. URL https://arxiv.org/abs/2405.21063.

Mingyu Shin and Soyi Jung. A survey of behavior tree-based task planning algorithms for autonomous robotic systems. In *2024 15th International Conference on Information and Communication Technology Convergence (ICTC)*, pages 2039–2041, 2024. doi: 10.1109/ICTC62082.2024.10827191.

Christopher Iliffe Sprague and Petter Ögren. Adding neural network controllers to behavior trees without destroying performance guarantees. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pages 3989–3996, 2022. doi: 10.1109/CDC51059.2022.9992501.

Jiankai Sun, Hao Sun, Tian Han, and Bolei Zhou. Neuro-symbolic program search for autonomous driving decision module design. In Jens Kober, Fabio Ramos, and Claire Tomlin, editors, *Proceedings of the 2020 Conference on Robot Learning*, volume 155 of *Proceedings of Machine Learning Research*, pages 21–30. PMLR, 16–18 Nov 2021. URL https://proceedings.mlr.press/v155/sun21a.html.

Hoang-Dung Tran, Diago Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T. Johnson. Star-based reachability analysis of deep neural networks. In Maurice H. ter Beek, Annabelle McIver, and José N. Oliveira, editors, *Formal Methods – The Next 30 Years*, pages 670–686, Cham, 2019. Springer International Publishing. ISBN 978-3-030-30942-8.

Qiang Wang, Huadong Dai, Yongxin Zhao, Min Zhang, and Simon Bliudze. Enabling behaviour tree verification via a translation to bip. In Diego Marmsoler and Meng Sun, editors, *Formal Aspects of Component Software*, pages 3–20, Cham, 2024. Springer Nature Switzerland. ISBN 978-3-031-71261-6.

Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J. Zico Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 29909–29921. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/fac7fead96dafceaf80c1daffeae82a4-Paper.pdf.

Yansong Wu, Fan Wu, Lingyun Chen, Kejia Chen, Samuel Schneider, Lars Johannsmeier, Zhenshan Bing, Fares J. Abu-Dakka, Alois Knoll, and Sami Haddadin. 1 khz behavior tree for self-adaptable tactile insertion. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 16002–16008, 2024. doi: 10.1109/ICRA57147.2024.10610835.