Safety Verification of Tree-Ensemble Policies via Predicate Abstraction

Chaahat Jain[a,*], **Lorenzo Cascioli**[b], **Laurens Devos**[b], **Marcel Vinzent**[a], **Marcel Steinmetz**[c], **Jesse Davis**[b] **and Jörg Hoffmann**[a,d]

[a]Saarland University, Saarland Informatics Campus, Germany
[b]Department of Computer Science, KU Leuven, Leuven, Belgium
[c]LAAS-CNRS, ANITI, Université de Toulouse, France
[d]German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany

Abstract. Learned action policies are gaining traction in AI, but come without safety guarantees. Recent work devised a method for safety verification of neural policies via predicate abstraction. Here we extend this approach to policies represented by tree ensembles, through replacing the underlying SMT queries with queries that can be dispatched by Veritas, a reasoning tool dedicated to tree ensembles. The query language supported by Veritas is limited, and we show how to encode richer constraints we need into additional trees and decision variables. We run experiments on benchmarks previously used to evaluate neural policy verification, and we design new benchmarks based on a logistics application at Airbus as well as on a real-world robotics domain. We find that (1) verification with Veritas vastly outperforms verification with Z3 and Gurobi; (2) tree-ensemble policies are much faster to verify than neural policies, while being competitive in policy quality; (3) our techniques are highly complementary to, and often outperform, an encoding of tree-ensemble policy verification into NUXMV.

1 Introduction

Learned action policies are gaining traction in AI [e.g., 30, 33, 34], including in AI planning [25, 22, 19, 40, 28, 35, 36]. However, such policies come without built-in safety guarantees. Given a policy π, a start condition ϕ_0, and an unsafety condition ϕ_u, how to verify whether an unsafe state $s^u \models \phi_u$ is reachable from a given start state $s^0 \models \phi_0$ under π? Research on this question is still in its early stages. In the formal methods community, prominent lines of works address neural controllers of dynamical systems [38, 41, 24, 17] or hybrid systems [26]. Here we follow up on recent work in the AI planning community [44, 45, 43] (henceforth VEA), which tackles neural policies taking discrete action choices in non-deterministic state spaces, via **policy predicate abstraction (PPA)**. PPA builds an over-approximating abstraction not of the full state space Θ, but of the state-space subgraph containing only those transitions taken by π. While building that abstraction, for each possible abstract state transition $(s_{\mathcal{P}}, l, s'_{\mathcal{P}})$, an SMT solver is queried to decide whether π selects the necessary label l for at least one corresponding concrete state transition (s, l, s'). This SMT query is dispatched to *Marabou* [29], an SMT solver tailored to neural network analysis.

Here we extend this approach to another family of learned action policies, namely ones represented by tree ensembles [7, 18, 11] which have proven to be powerful predictors in many ML problems [21]. Our key observation is that PPA is agnostic to the policy representation, provided there exists a reasoning mechanism to which the abstract state transition queries can be dispatched. Given this, we can leverage progress on reasoning methods tailored to ML models other than neural networks. Here, we show how to do this for the **Veritas** tool, that excels in analyzing tree-ensemble predictions [15, 9]. We thus establish the first machinery dedicated to safety verification of tree-ensemble policies.

A key challenge in using Veritas within PPA is the expressivity of constraints needed. The query language supported by Veritas (in its state-of-the-art implementation for multiclass classifiers) is limited to box constraints. Yet we require support for general linear constraints, and for disjuntions thereof. We show how to encode these richer constraints into additional trees that are added to the policy ensemble.

In particular, we address applicability filtering, which selects the best-predicted-value action among only the applicable actions in a state (rather than among all actions), thus ensuring that the policy never selects inapplicable actions. This adds complexity to PPA verification because the abstract transition queries now involve large subformulas checking action applicability. Recent work has shown how to tackle this for neural policies, through SMT formula transformation and simplification [43]. Here we show how to perform applicability filtering for tree-ensemble policies by constructing trees that penalize inapplicable actions. The applicability conditions in this context are in disjunctive normal form (DNF). A naïve encoding of a DNF as a decision tree explodes exponentially due to the replication problem [31]. Instead, we break the formula similar to Tseytin Transformations [42] by introducing auxiliary variables.

As a side contribution of our work, we show that PPA – for both, neural and tree-ensemble policies – can be successfully applied to inifinite state spaces. Previously, PPA was applied only to problems with bounded-integer state variables. Here we include a case study (Turtlebot, see below) with real-valued state variables. We tackle both kinds of state variables throughout. Continuous state variables do not require any changes to PPA per se, but constitute an addtional challenge in encoding linear constraints for Veritas.

We run experiments on VEA's benchmarks, as well as on two new

[*] Corresponding Author. Email: jain@cs.uni-saarland.de

benchmarks we contribute here, called Beluga and Turtlebot. Beluga abstractly encodes a planning application at Airbus, namely a logistics problem involving the transport of airplane parts between Beluga transport planes and a factory, using a particular kind of racks for intermediate storage. Turtlebot is a case study in a simple robotics domain, adapted from prior work [1] on purely SMT/Marabou-based verification in that domain. Specifically, we consider the neural Turtlebot-control policies trained by these authors. We extend the very basic safety property they verified (will the robot ever decide to move into a wall it stands directly in front of?) to a scalable property (will the robot ever move into a wall it stands k steps in front of?). In all benchmarks, we use neural policies as teachers for imitation learning of tree-ensemble policies. We empirically compare our approach to competing methods in three ways:

(1) To evaluate the benefit of using Veritas to dispatch the abstract transition tests in PPA, we compare to a version of PPA using instead **Z3** [14], and a version using instead **Gurobi** [23] (with straightforward SMT/MILP encodings respectively).
(2) To evaluate the merits of neural vs. tree-ensemble policy representations, we compare policy quality and verifiability of our tree ensembles vs. their neural teachers.
(3) While ours is the first work dedicated to verification of tree-ensemble policies, one can encode such policies into standard verification languages, and thus apply standard verification methods not tailored to ML models. To evaluate this alternative, we compare PPA to a broad range of state-of-the-art verification methods implemented in NUXMV [10].

Our findings are as follows. (1) PPA with Z3 or Gurobi is outperformed everywhere except our smallest benchmark instances, decisively showing the benefit of using the tailored solver Veritas instead. (2) Tree ensemble policies are typically orders of magnitude faster to verify than neural policies, while (in our benchmarks) being able to learn policies of similar quality as their neural teachers; this suggests a better trade-off between policy quality vs. verifiability. (3) PPA is highly complementary to NUXMV, and outperforms NUXMV in many cases. In particular, PPA's closest relative, predicate abstraction as implemented in NUXMV, is unable to tackle any of our benchmark instances, so is outclassed completely.

2 Background

2.1 *Policy Safety Verification and PPA*

We consider transition systems described by a tuple $\langle \mathcal{V}, \mathcal{L}, \mathcal{O} \rangle$. $\mathcal{V}$ is a finite set of state variables. For each $v \in \mathcal{V}$ the domain D_v is a bounded integer or a bounded real-valued interval. $\mathcal{L}$ is a finite set of **action labels**. $\mathcal{O}$ is a finite set of **operators**. Φ denotes the set of **linear constraints** over $\mathcal{V}$, i.e., of the form $\sum_{v \in \mathcal{V}} d_v \cdot v \bowtie c$ with rational coefficients d_v and c, $\bowtie \in \{\leq, =, \geq\}$, and Boolean combinations thereof. An **operator** $o \in \mathcal{O}$ is a tuple (g, l, u) with **label** $l \in \mathcal{L}$, **guard** $g \in \Phi$ and **update** u over $\mathcal{V}$, where $u(v)$ is either a linear assignment $\sum_{v \in \mathcal{V}} d_v \cdot v + c$ or a bounded interval $[u_v^l, u_v^u]$.

The **state space** of $\langle \mathcal{V}, \mathcal{L}, \mathcal{O} \rangle$ is a transition system $\Theta = \langle \mathcal{S}, \mathcal{L}, \mathcal{T} \rangle$. The set of **states** $\mathcal{S}$ is the set of complete variable assignments over $\mathcal{V}$. The set of **transitions** $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{L} \times \mathcal{S}$ contains (s, l, s') iff there exists an operator $o = (g, l, u)$ such that $g(s)$ evaluates to true, also written $s \models g$, and, for each $v \in \mathcal{V}$, $s'(v) = u(v)(s)$ for assignments and $s'(v) \in [u_v^l, u_v^u]$ for interval udaptes, also abbreviated $s' \models u(s)$. Θ is finite if all variables have integer domains, but infinite otherwise. For a state s, we denote with $\mathcal{L}(s) = \{l \in \mathcal{L} \mid \exists s' \in \mathcal{S} \colon (s, l, s') \in \mathcal{T}\}$ the set of actions **applicable** in s. We assume, w.l.o.g., that $\mathcal{L}(s) \neq \emptyset$ for all states s.

An **action policy** π is a function $\mathcal{S} \to \mathcal{L}$. We consider action policies that are represented by a value function $Q \colon \mathcal{S} \times \mathcal{L} \mapsto \mathbb{R}$, defining $\pi_Q(s) := argmax_{l \in \mathcal{L}} Q(s, l)$, i.e., the policy selects the highest ranked action in s. Alternatively, we consider **applicability filtering** where $\pi_Q^{\text{app}}(s) := argmax_{l \in \mathcal{L}(s)} Q(s, l)$, constraining the selection to applicable actions. A policy π induces the **policy graph** $\Theta^\pi = (\mathcal{S}, \mathcal{L}, \mathcal{T}^\pi)$ over the same states and action labels as Θ, and $\mathcal{T}^\pi \subseteq \mathcal{T}$ where $(s, l, s') \in \mathcal{T}^\pi$ iff $\pi(s) = l$.

A **reach-avoid property** is a tuple $(\phi_0, \phi_u, \mathcal{G})$, where $\phi_0 \in \Phi$ identifies the start states, $\phi_u \in \Phi$ identifies the unsafe states and $\mathcal{G} \in \Phi$ identifies the goal states. A policy π is **unsafe** with respect to $(\phi_0, \phi_u, \mathcal{G})$ iff there exists a path $\langle s^0, \pi(s^0), s^1, \ldots, \pi(s^{n-1}), s^n \rangle$ in Θ^π such that $s^0 \models \phi_0$, $s^n \models \phi_u$, and $s^i \not\models \mathcal{G}$ for all $i \in \{0, \ldots, n-1\}$. π is **safe** if no such path exists. In words, the policy's task is to *reach* the goal $\mathcal{G}$ while *avoiding* unsafe states ϕ_u.

Policy predicate abstraction (PPA) [44] is a technique for verifying a policy's safety by *abstracting* the the graph Θ^π induced by π (as opposed to abstracting the the full state space Θ as done in classical predicate abstraction [20]). Given a set of **predicates** $\mathcal{P} \subseteq \Phi$, an **abstract state** $s_\mathcal{P}$ is a (complete) truth value assignment $\mathcal{P}$. $[s_\mathcal{P}] = \{s \in \mathcal{S} \mid \forall p \in \mathcal{P} \colon s \models p \text{ iff } s_\mathcal{P}(p) = \top\}$ denotes the set of all concrete states represented by $s_\mathcal{P}$. The **policy predicate abstraction** of Θ over $\mathcal{P}$ and π is the labeled transition system $\Theta_\mathcal{P}^\pi = \langle \mathcal{S}_\mathcal{P}, \mathcal{L}, \mathcal{T}_\mathcal{P}^\pi \rangle$ where $\mathcal{S}_\mathcal{P}$ is the set of all abstract states over $\mathcal{P}$ and $(s_\mathcal{P}, l, s'_\mathcal{P}) \in \mathcal{T}_\mathcal{P}^\pi$ iff there exist $s \in [s_\mathcal{P}]$ and $s' \in [s'_\mathcal{P}]$ such that $(s, l, s') \in \mathcal{T}^\pi$.

Note that $\Theta_\mathcal{P}^\pi$ is by definition guaranteed to be finite. $\Theta_\mathcal{P}^\pi$ is **unsafe** if there exists an abstract path $\langle s_\mathcal{P}^0, l^1, \ldots, l^n, s_\mathcal{P}^n \rangle$ in $\Theta_\mathcal{P}^\pi$ such that $s_\mathcal{P}^0 \models \phi_0$, $s_\mathcal{P}^n \models \phi_u$, and $s_\mathcal{P}^i \models \neg\mathcal{G}$ for $i \in \{0, \ldots, n-1\}$, where $s_\mathcal{P} \models \phi$ iff $\exists s \in [s_\mathcal{P}] \colon s \models \phi$. Otherwise $\Theta_\mathcal{P}^\pi$ is **safe**. If $\Theta_\mathcal{P}^\pi$ is safe, then π is safe as well. Vice versa, however, an unsafe path in $\Theta_\mathcal{P}^\pi$ may be **spurious**, i.e., there might be no corresponding path in Θ^π. PPA with **counterexample-guided abstraction refinement** (CEGAR) [45] iteratively removes such spurious abstract paths by refining $\mathcal{P}$, until either $\Theta_\mathcal{P}^\pi$ is proven safe, or a non-spurious abstract path is found showing that π is unsafe.

Constructing $\Theta_\mathcal{P}^\pi$ requires solving the **abstract transition problem**, i.e., the problem of, given an abstract-state-label triple $(s_\mathcal{P}, l, s'_\mathcal{P})$, deciding whether $(s_\mathcal{P}, l, s'_\mathcal{P}) \in \mathcal{T}_\mathcal{P}^\pi$. As per the definition of $\Theta_\mathcal{P}^\pi$, this is the case iff for some operator (g, l, u) there exist concrete states $s \in [s_\mathcal{P}]$ and $s' \in [s'_\mathcal{P}]$ such that $s \models g$, $s' \models u(s)$ and $\pi(s) = l$. In classical predicate abstraction where no policy is considered and, hence, the condition $\pi(s) = l$ is dropped, such abstract transition problems are routinely encoded into satisfiability modulo theories (SMT) [4]. The key challenge for PPA is how to encode this policy selection condition. VEA [44] have addressed this problem for action policies π_Q where Q is given by a feed forward neural network, making use of an SMT solver specialized to neural networks [29]. VEA [43] recently extended this approach to handling neural policies π_Q^{app} under applicability filtering.

In this work we extend the PPA framework to the verification of tree-ensemble policies. The key issue is which solver to use to dispatch abstract transition problem queries.

2.2 *Additive Tree Ensembles*

Given an input space $\mathcal{X} \subseteq \mathbb{R}^k$, a **binary decision tree** T consists of two types of nodes. *Internal nodes* store a reference to a left and right sub-tree, and a branching condition $v_i < \alpha$, where v_i is an attribute of the input $\vec{v} \in \mathcal{X}$ and $\alpha \in \mathbb{R}$ is a constant split value. *Leaf nodes* have no children and store a leaf value $\nu \in \mathbb{R}$. For an input $\vec{v} \in \mathcal{X}$, $T(\vec{v})$ is the value of the unique leaf reached by traversing the tree, following the left child of an inner node if the branching condition is satisfied by $\vec{v}$ and the right child otherwise. Each leaf l is associated to a $box(l)$, which defines an hypercube of the input space by conjoining all the split conditions of the internal nodes encountered in the root-to-leaf path. All examples $\vec{v} \in box(l)$ evaluate to the value of l. An **additive tree ensemble** (tree ensemble for short) is a sum of binary trees $\boldsymbol{T}(\vec{v}) := \sum_{i=1}^{n} T^i(\vec{v})$. We consider multi-class classifiers $\mathcal{C}_{\boldsymbol{T}}$ which associate each class C with a tree ensemble $\boldsymbol{T}_C$ and define the class predictions through $\mathcal{C}(\vec{v}) := argmax_C \boldsymbol{T}_C(\vec{v})$ [11, 7].

2.3 *The Veritas Tool*

Given a classifier $\mathcal{C}$, an important problem is determining whether an input exists such that $\mathcal{C}(\vec{v}) = C^t$ for a target class C^t. Typically, the input must belong to a constrained region Γ of the input space. **Veritas**[1] is a state-of-the-art tool for tree-ensembles [15, 9], which casts this decision problem as the optimization problem

$$\begin{aligned} &\max_{\vec{v}} f^*(\vec{v}) \qquad \text{subject to} \quad \vec{v} \models \Gamma, \\ &f^*(\vec{v}) := \left[\boldsymbol{T}_{C^t}(\vec{v}) - \max_{C \neq C^t} \boldsymbol{T}_C(\vec{v}) \right] \end{aligned} \tag{1}$$

Veritas requires Γ to be a conjunction of **box constraints**, i.e., conditions $l_{v_i} \leq v_i < u_{v_i}$ where $l_{v_i}, u_{v_i} \in \mathbb{R} \cup \{-\infty, \infty\}$. It solves (1) by means of a heuristic search that incrementally refines the starting box Γ to a solution that is still a box, but only overlaps with one leaf of each tree in the ensemble. If a solution with positive objective value exists for (1), then class C^t will be selected over all other possible classes for every input within the solution box.

3 Encoding PPA Queries as Box Constraints

Our key observation is that PPA is agnostic to the policy representation. To verify a policy represented by a tree ensemble, all we require is a solver to which the abstract transition problem queries can be dispatched. Provided the solver is correct in answering these queries, the correctness of the overall PPA machinery is preserved.

General purpose constraint solvers like Z3 [14] or Gurobi [23] fit this profile. Here we instead investigate how to use Veritas, which is tailored to tree ensembles, in PPA. As our experiments show, this yields much superior verification performance.

The key challenge here is the expressivity of constraints needed. As discussed, Veritas requires the input constraint to be a conjunction of box constraints. Yet PPA requires complex constraints built from linear arithmetic. Our core technical contribution is how to encode these constraints for effective use of Veritas in the PPA context.

Our discussion is organized as follows. In Section 4, we show that the PPA abstract transition queries can take the form *(*)* $\exists \vec{v}\colon \vec{v} \models \Gamma \wedge \Psi$ *and* $\mathcal{C}_{\boldsymbol{T}}(\vec{v}) = C^t$ where Γ is a conjunction of box constraints and Ψ is a Boolean combination of linear constraints. Here, we show that such queries can be encoded as $\exists \vec{v}'\colon \vec{v}' \models \Gamma$ *and* $\mathcal{C}_{\boldsymbol{T}'}(\vec{v}') = C^t$ where $\boldsymbol{T}'$ penalizes inputs violating Ψ. We proceed bottom up, first considering the encoding of individual linear constraints, then detailing how to efficiently deal with their Boolean combination.

[1] https://github.com/laudv/veritas

3.1 *Linear Constraints over Integer Variables*

Let $\phi := d_1 v_1 + \ldots d_r v_r \bowtie c$ be a linear constraint. The key observation is that the query (*) for $\Psi = \phi$ can be reduced to pure box constraints by manipulating the tree ensemble. More specifically, ϕ can be converted into an additional tree ensemble $\boldsymbol{T}_\phi$ such that $\exists \vec{v}\colon \vec{v} \models \Gamma \wedge \phi$ and $\mathcal{C}_{\boldsymbol{T}}(\vec{v}) = C^t$ iff $\exists \vec{v}'\colon \vec{v}' \models \Gamma$ and $\mathcal{C}_{\boldsymbol{T}'}(\vec{x}') = C^t$ for the modified tree-ensemble classifier $\mathcal{C}_{\boldsymbol{T}'}$ in which the tree ensemble for C^t, $\boldsymbol{T}_{C^t}$, is replaced by the union with $\boldsymbol{T}_\phi$ (yielding the sum $\boldsymbol{T}_{C^t} + \boldsymbol{T}_\phi$). The latter query can then be handled by Veritas. We next show how to construct $\boldsymbol{T}_\phi$ for *integer variables* $v_1, \ldots, v_r$.

Let $\nu_{\max}^T$ denote the maximal leaf value in a binary tree T, $\nu_{\min}^T$ denote the minimal leaf value, and let $\nu_{\max}^{\boldsymbol{T}} := \max_C \sum_{T \in \boldsymbol{T}_C} \nu_{\max}^T$ and similarly $\nu_{\min}^{\boldsymbol{T}}$ be upper and lower bounds on the maximal and minimal possible value of all the tree ensembles. Finally, let

$$\delta := \nu_{\min}^{\boldsymbol{T}} - \nu_{\max}^{\boldsymbol{T}} \tag{2}$$

be the difference between the minimal and maximal tree-ensemble values. To generate the desired tree ensemble $\boldsymbol{T}_\phi$, we first enumerate all assignments to $v_1, \ldots, v_r$ violating $d_1 v_1 + \ldots d_r v_r \bowtie c$. This is possible as the domains of the integer variables is bounded (hence finite). However, it is exponential in the number of variables r in the constraint, which limits scalability in general, but works for small r as is typically the case in the PPA setting, which is our primary interest. For each violating assignment α, we then add to $\boldsymbol{T}_\phi$ a binary tree T^α that evaluates to $T^\alpha(\vec{v}) = \delta$ if $\vec{v}$ agrees with α, and $T^\alpha(\vec{v}) = 0$ otherwise. An example for $\phi := y - x = 3$ with the violating assignment $\alpha : x = 1, y = 3$ is shown in Fig. 1.

Given that every $\vec{v}$ can agree with at most one violating assignment α, we have $\boldsymbol{T}_\phi(\vec{v}) \in \{0, \delta\}$, and $\boldsymbol{T}_\phi(\vec{v}) = 0$ iff $\vec{v} \models \phi$ holds by construction. This means in particular that $(\boldsymbol{T}_{C^t} + \boldsymbol{T}_\phi)(\vec{v}) = \boldsymbol{T}_{C^t}(\vec{v})$ for all $\vec{v}$ s.t. $\vec{v} \models \phi$. In other words, it holds for every $\vec{v}$ where $\mathcal{C}_{\boldsymbol{T}}(\vec{v}) = C^t$ and $\vec{v} \models \phi$ that $\mathcal{C}_{\boldsymbol{T}'}(\vec{v}) = C^t$. Vice versa, for all $\vec{v} \not\models \phi$ and all classes $C \neq C^t$, the choice of δ ensures that

$$\begin{aligned} (\boldsymbol{T}_{C^t} + \boldsymbol{T}_\phi)(\vec{v}) &= \boldsymbol{T}_{C^t}(\vec{v}) + \delta \\ &= \boldsymbol{T}_{C^t}(\vec{v}) - \nu_{\max}^{\boldsymbol{T}} + \nu_{\min}^{\boldsymbol{T}} \\ &\leq \nu_{\min}^{\boldsymbol{T}} \leq \boldsymbol{T}_C(\vec{v}) \end{aligned}$$

and therewith $\mathcal{C}_{\boldsymbol{T}'}(\vec{v}) \neq C^t$. Combining both observations yields:

Theorem 1. *It holds for every tree-ensemble classifier $\mathcal{C}_{\boldsymbol{T}}$, every conjunction of box constraints Γ, every linear constraint ϕ over integer variables, and every target class C^t that $\exists \vec{v} \models \Gamma \wedge \phi$ with $\mathcal{C}_{\boldsymbol{T}}(\vec{v}) = C^t$ iff $\exists \vec{v}' \models \Gamma$ with $\mathcal{C}_{\boldsymbol{T}'}(\vec{v}') = C^t$ where $\mathcal{C}_{\boldsymbol{T}'}$ replaces $\boldsymbol{T}_{C^t}$ with $\boldsymbol{T}_{C^t} + \boldsymbol{T}_\phi$.*

The compilation can be extended to conjunctions $\phi_1 \wedge \cdots \wedge \phi_n$ of linear constraints, using the observation that $\sum_{i=1}^{n} \boldsymbol{T}_{\phi_i}(\vec{v}) \leq \delta$ as soon as $\vec{v} \not\models \phi_i$ for any $i \in \{1, \ldots, n\}$; while being 0 if all constraints are satisfied. Hence, substituting the target tree ensemble $\boldsymbol{T}_{C^t}$ by the union with the tree ensembles $\boldsymbol{T}_{\phi_1}, \ldots, \boldsymbol{T}_{\phi_n}$ yields:

Corollary 2. *It holds for every tree-ensemble classifier $\mathcal{C}_{\boldsymbol{T}}$, every conjunction of box constraints Γ, linear constraints $\phi_1, \ldots, \phi_n$ over integer variables, and every target class C^t that $\exists \vec{v} \models \Gamma \wedge \bigwedge_{i=1}^{n} \phi_i$ with $\mathcal{C}_{\boldsymbol{T}}(\vec{v}) = C^t$ iff $\exists \vec{v}' \models \Gamma$ with $\mathcal{C}_{\boldsymbol{T}'}(\vec{v}') = C^t$ where $\mathcal{C}_{\boldsymbol{T}'}$ replaces $\boldsymbol{T}_{C^t}$ with $\boldsymbol{T}_{C^t} + \sum_{i=1}^{n} \boldsymbol{T}_{\phi_i}$.*

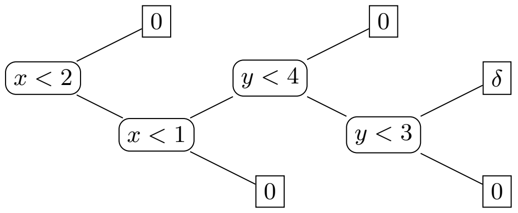

Figure 1. Depiction of a tree T^{α}, for linear constraint $y - x = 3$ and violating assignment $\alpha : x = 1, y = 3$.

3.2 Linear Constraints over Real-Valued Variables

Let $\phi := d_1 v_1 + \dots d_r v_r \bowtie c$ again be a linear constraint. The construction of $\boldsymbol{T}_\phi$ just sketched works only if the domains of all variables appearing in ϕ are finite. To handle (bounded) real-valued variables, we exploit the structure of the tree ensembles themselves to identify the values that need to be distinguished. To this end, for an input variable v and a binary decision tree T, we denote with $D_v(T)$ the set of values α over all branching conditions $v < \alpha$ in T. For a tree ensemble $\boldsymbol{T}$, $D_v(\boldsymbol{T}) := \bigcup_{T \in \boldsymbol{T}} D_v(T)$ is the union of the branching values for v in all the trees in $\boldsymbol{T}$, and for the classifier $\mathcal{C}_{\boldsymbol{T}}$, $D_v(\mathcal{C}_{\boldsymbol{T}}) := \bigcup_C D_v(\boldsymbol{T}_C)$ gives the union over the tree ensembles of all classes. Finally, for a variable v and set of values D_x, we denote with $\mathbf{B}[D_x] = \{l \le v < u \mid l, u \in D_x \cup \{-\infty, \infty\} : l < u\}$ the set of box constraints on v induced by D_x.

We encode ϕ as the following tree ensemble $\boldsymbol{T}_\phi^{\mathbf{B}}$. For each combination of box constraints $\vec{\beta} = \langle \beta_1, \dots, \beta_r \rangle \in \mathbf{B}[D_{v_1}(\mathcal{C}_{\boldsymbol{T}})] \times \dots \times \mathbf{B}[D_{v_r}(\mathcal{C}_{\boldsymbol{T}})]$ such that $\phi \wedge \Gamma \wedge \bigwedge_{i=1}^{r} \beta_i$ is unsatisfiable (which can be checked via SMT), we add to $\boldsymbol{T}_\phi^{\mathbf{B}}$ the tree $T_{\vec{\beta}}$ that evaluates to $T_{\vec{\beta}}(\vec{v}) = \delta$ (cf. Eq. (2)) if $\vec{v} \models \bigwedge_{i=1}^{r} \beta_i$ and $T_{\vec{\beta}}(\vec{v}) = 0$ otherwise.

Example 1. *Consider the linear constraint $\phi := y - x = 1$, and suppose that $D_x(\mathcal{C}_{\boldsymbol{T}}) = \{1.3\}$, $D_y(\mathcal{C}_{\boldsymbol{T}}) = \{2.4\}$. Here, $\mathbf{B}[D_x(\mathcal{C}_{\boldsymbol{T}})] = \{-\infty \le x < 1.3, 1.3 \le x < \infty\}$ and $\mathbf{B}[D_y(\mathcal{C}_{\boldsymbol{T}})] = \{-\infty \le y < 2.4, 2.4 \le y < \infty\}$. Consider $\vec{\beta} = \langle \beta_x, \beta_y \rangle$ with $\beta_x := -\infty \le x < 1.3$ and $\beta_y := 2.4 \le y < \infty$. Obviously, $\phi \wedge \beta_x \wedge \beta_y$ is unsatisfiable. The tree $T_{\vec{\beta}}$ is shown in Fig. 2.*

This construction guarantees for all $\vec{v}$ where $T_{\vec{\beta}}(\vec{v}) = \delta$ that $\vec{v} \not\models \phi$. In other words, if $\boldsymbol{T}_\phi(\vec{v}) \le \delta$ then $\vec{v} \not\models \phi$. Conversely, however, $\boldsymbol{T}_\phi(\vec{v}) = 0$ does not imply in general that $\vec{v} \models \phi$ as opposed to our prior linear-constraint tree encoding. Nevertheless, $\boldsymbol{T}_\phi^{\mathbf{B}}$ suffices for our purposes.

Namely, let $\mathcal{C}_{\boldsymbol{T}'}$ again be the tree ensemble classifier where we substitute $\boldsymbol{T}_{C^t}$ by $\boldsymbol{T}_{C^t} + \boldsymbol{T}_\phi^{\mathbf{B}}$. We want to show that $\exists \vec{v} \colon \vec{v} \models \Gamma \wedge \phi$ such that $\mathcal{C}_{\boldsymbol{T}}(\vec{v}) = C^t$ iff $\exists \vec{v}' \colon \vec{v}' \models \Gamma$ and $\mathcal{C}_{\boldsymbol{T}'}(\vec{v}') = C^t$. The only if case follows directly from our previous observation. Specifically, with $\vec{v} \models \phi$, we have $\boldsymbol{T}_\phi^{\mathbf{B}}(\vec{v}) = 0$, which implies $\mathcal{C}_{\boldsymbol{T}'}(\vec{v}) = \mathcal{C}_{\boldsymbol{T}}(\vec{v}) = C^t$. Consider the opposite case. Suppose for contradiction that there were no $\vec{v}$ satisfying $\vec{v} \models \Gamma \wedge \phi$ so that $\mathcal{C}_{\boldsymbol{T}}(\vec{v}) = C^t$. Consider the box-constraint combination $\beta = \langle \beta_1, \dots, \beta_r \rangle$ derived from $\vec{v}'$ as follows. For each index $i \in \{1, \dots, r\}$, let β_i be the box constraint $l_i \le v_i < u_i$ where $l_i := \sup\{d \in D_{v_i}(\mathcal{C}_{\boldsymbol{T}}) \mid d \le v_i'\}$ is the largest branch value lower bounding the corresponding value in $\vec{v}'$, and, similarly, $u_i := \inf\{d \in D_{v_i}(\mathcal{C}_{\boldsymbol{T}}) \mid d > v_i'\}$. The selection of $\vec{\beta}$ ensures that every $\vec{v}''$ so that $l_i \le v_i'' < u_i$, for all $i \in \{1, \dots, r\}$, and which agrees with $\vec{v}'$ on the values at the remaining indices, falls into the same leaf box $box(l)$ for every tree in the ensemble. Therefore, $\mathcal{C}_{\boldsymbol{T}'}(\vec{x''}) = \mathcal{C}_{\boldsymbol{T}'}(\vec{x'}) = C^t$. Since $\boldsymbol{T}_{C^t}(\vec{v}'') \ge (\boldsymbol{T}_{C^t} + \boldsymbol{T}_\phi^{\mathbf{B}})(\vec{v}'')$ as per the definition of $\boldsymbol{T}_\phi^{\mathbf{B}}$, it follows that $\mathcal{C}_{\boldsymbol{T}}(\vec{v}'') = C^t$, which, by assumption, means that every such $\vec{v}''$ must violate $\phi \wedge \Gamma$. Hence, $T_{\vec{\beta}}(\vec{v}'') = \delta$. It follows that $(\boldsymbol{T}_{C^t} + \boldsymbol{T}_\phi^{\mathbf{B}})(\vec{v}'') \le \boldsymbol{T}_{C^t}(\vec{v}'') + \delta \le \boldsymbol{T}_C(\vec{v}'')$ for all classes C, so, in particular, $(\boldsymbol{T}_{C^t} + \boldsymbol{T}_\phi^{\mathbf{B}})(\vec{v}') \le \boldsymbol{T}_C(\vec{v}')$, in contradiction to $\mathcal{C}_{\boldsymbol{T}'}(\vec{v}') = C^t$. The encoding can be extended to conjunctions of linear constraints in the same manner as described above. We get:

Theorem 3. *It holds for every tree-ensemble classifier $\mathcal{C}_{\boldsymbol{T}}$, every conjunction of box constraints Γ, arbitrary linear constraints $\phi_1, \dots, \phi_n$, and every target class C^t that $\exists \vec{v} \models \Gamma \wedge \bigwedge_{i=1}^{n} \phi_i$ with $\mathcal{C}_{\boldsymbol{T}}(\vec{v}) = C^t$ iff $\exists \vec{v}' \models \Gamma$ with $\mathcal{C}_{\boldsymbol{T}'}(\vec{v}') = C^t$ where $\mathcal{C}_{\boldsymbol{T}'}$ replaces $\boldsymbol{T}_{C^t}$ with $\boldsymbol{T}_{C^t} + \sum_{i=1}^{n} \boldsymbol{T}_{\phi_i}^{\mathbf{B}}$.*

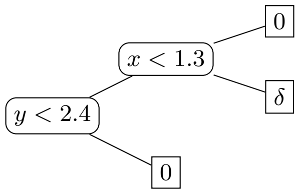

Figure 2. Encoding $T_{\vec{\beta}}$ for linear constraint $\phi : y = x + 1$, for $\vec{\beta} = \langle -\infty \le x \le 1.3, 2.4 \le y \le \infty \rangle$. Here, $x \le 1.3 \wedge y \ge 2.4 \wedge \phi$ is unsatisfiable. Here ϵ accounts for floating point precision.

Note that the construction of $\boldsymbol{T}_\phi^{\mathbf{B}}$ is not limited to real-valued variables, but works for integer variables as well. However, the construction $\boldsymbol{T}_\phi^{\mathbf{B}}$ is in general more complex than that of $\boldsymbol{T}_\phi$ for integer variables; the former scaling exponentially in r and quadratically in the variable domain size, while the latter also scales exponentially in r but only linearly in the domain size.

3.3 Disjunctive Constraints

The query (*) for a constraint $\Gamma \wedge \Psi$ where Ψ is a disjunction of linear constraints $\bigvee_{i=1}^{n} \phi_i$ can be cast as the separate queries of n box-constraint tasks, using the previously introduced technique to deal with the linear constraints. However, this approach does not scale to conjunctions $\Psi = \bigwedge_{i=1}^{m} \psi_m$ of disjunctions (or symmetrically, disjunctions of conjunctions), which would decompose into n^m prue box-constraint queries. Here we show how to encode such constraints *efficiently*, breaking such Ψ down into simpler formulae following ideas similar to the Tseytin transformation [42].

We modify the tree ensemble classifier, introducing auxiliary integer input variables v_ϕ, for each linear constraint ϕ in Ψ, meant to represent the satisfaction of ϕ. Similar to before, we enforce their semantic meaning by including additional decision trees in the ensembles. These helper variables will be only used in the trees we add to the ensemble; the original ones are left untouched. For simplicity, we assume here that all linear constraints are over integer variables, allowing us to use the $\boldsymbol{T}_\phi$ encoding from Section 3.1. The overall construction can however be straightforwardly generalized to any combination of linear constraints, including combinations of integer-only and real-valued-variable linear constraints.

Let ϕ be some linear constraint in Ψ. Let $\boldsymbol{T}_\phi$ be the tree ensemble representing the dissatisfaction of ϕ, i.e., the ensemble guaranteeing $\boldsymbol{T}_\phi(\vec{v}) = \delta$ iff $\vec{v} \not\models \phi$, and $\boldsymbol{T}_\phi(\vec{v}) = 0$ otherwise. To link the auxiliary variable v_ϕ with the satisfaction of ϕ, we modify $\boldsymbol{T}_\phi$, adding to all decision trees the additional branching condition $\mathsf{v}_\phi < 1$. As a result, the tree ensemble evaluates to δ for the extended input $[\vec{v}\ \mathsf{v}_\phi]$ only if ϕ is violated by $\vec{v}$ while $\mathsf{v}_\phi < 1$. Vice versa, if the tree evaluates to 0 and $\mathsf{v}_\phi < 1$, then $\vec{v}$ must necessarily satisfy ϕ. In other words,

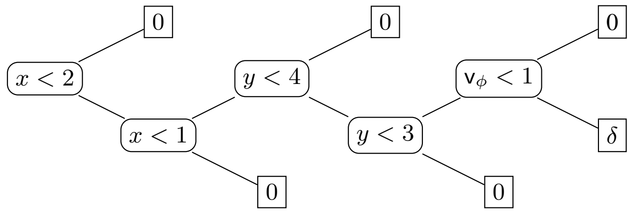

Figure 3. Extension of the tree T^α from Fig. 1 by the variable v_ϕ.

$\mathsf{v}_\phi < 1$ acts as an indicator of the satisfaction of ϕ. Fig. 3 shows the extension of the example from Fig. 1. Let $\boldsymbol{T}_\Psi$ be the union over the so-obtained tree ensembles for all constraints in Ψ. Finally, we represent each disjunction ψ_i in Ψ as a single decision tree, T_{ψ_i}, which evaluates to 0 iff $\mathsf{v}_\phi < 1$ holds for any linear constraint in ψ_i, and to δ otherwise. Fig. 4 shows an example.

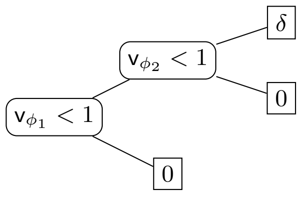

Figure 4. Encoding T_ψ for a formula $\psi = \phi_1 \vee \phi_2$. Here v_ϕ is an indicator variable to represent the dissatisfaction of linear constraint ϕ.

Let $\vec{v}^*$ denote a feature vector for the extended tree ensemble input space. Suppose that both $\boldsymbol{T}_\Psi(\vec{v}^*) = 0$ and $\sum_{i=1}^n T_{\psi_i}(\vec{v}^*) = 0$. The latter implies that every disjunction ψ_i contains a linear constraint ϕ such that $\mathsf{v}_\phi < 1$. The former then implies, as noted before, that $\vec{v}^*$ satisfies ϕ. In combination, $\vec{v}^*$ therefore satisfies all the disjunctions ψ_i. This allows to cast the query over the constraint $\Gamma \wedge \Psi$, where $\Psi = \bigwedge_{i=1}^n \psi_i$, as the query $\exists \vec{v}^* \colon \vec{v}^* \models \Gamma$ and $\mathcal{C}_{\boldsymbol{T}'}(\vec{v}^*) = C^t$ for the modified classifier $\mathcal{C}_{\boldsymbol{T}'}$ where the tree ensemble $\boldsymbol{T}_{C^t}$ is replaced by $\boldsymbol{T}_{C^t} + \boldsymbol{T}_\Psi + \sum_{i=1}^n T_{\psi_i}$. This is true because, if the original query has a solution $\vec{v}$, one obtains a solution $\vec{v}^*$ to the compiled query simply by assigning the auxiliary variables values according to the satisfaction of the linear constraints. On the other hand, if $\vec{v}^*$ is a solution to the compiled query, it immediately follows that $\boldsymbol{T}_\Psi(\vec{v}^*) = 0$ and $\sum_{i=1}^n T_{\psi_i}(\vec{v}^*) = 0$, which as just shown implies that $\vec{v}^*$ satisfies $\Gamma \wedge \Psi$, and therefore gives rise to a solution to the original query where one simply drops the auxiliary variables.

Theorem 4. *It holds for every tree-ensemble classifier $\mathcal{C}_{\boldsymbol{T}}$, every conjunction of box constraints Γ, arbitrary disjunctions over linear constraints $\psi_1, \ldots, \psi_n$, and every target class C^t that $\exists \vec{v} \models \Gamma \wedge \bigwedge_{i=1}^n \psi_i$ with $\mathcal{C}_{\boldsymbol{T}}(\vec{v}) = C^t$ iff $\exists \vec{v}^* \models \Gamma$ with $\mathcal{C}_{\boldsymbol{T}'}(\vec{v}^*) = C^t$ where $\mathcal{C}_{\boldsymbol{T}'}$ replaces $\boldsymbol{T}_{C^t}$ with $\boldsymbol{T}_{C^t} + \boldsymbol{T}_\Psi + \sum_{i=1}^n T_{\psi_i}$.*

Note that this construction can be extended to arbitrarily nested Boolean combinations of linear constraints, by introducing also auxiliary variables v_ψ representing sub-formulas ψ, and linking them to the satisfaction of ψ via additional decision trees similar to those for disjunctions described above.

4 Policy Predicate Abstraction for Tree Ensembles

Given the tools from Section 3, we are finally ready to detail how to verify the safety of tree-ensemble action policies. We consider policies represented by tree-ensemble classifiers $\mathcal{C}$ that predict an action label from $\mathcal{L}$ (the classes), taking the state variables as input. More specifically, as noted before, we consider two kinds of policies, $\pi_{\mathcal{C}}(s) = \mathcal{C}(s) = argmax_{l \in \mathcal{L}} \boldsymbol{T}_l(s)$, using directly the predictions of the classifier, respectively $\pi_{\mathcal{C}}^{\text{app}}(s) = argmax_{l \in \mathcal{L}(s)} \boldsymbol{T}_l(s)$, where we restrict classification to the applicable actions only. To verify the safety of these policies, we observe that PPA in principle applies to arbitrary policy representations, provided that a method solving the abstract policy-transition problem is available. Here we show how to realize this potential for tree-ensemble policies.

Let $\mathcal{P} \subseteq \Phi$ be a set of predicates, and let $(s_{\mathcal{P}}, l, s'_{\mathcal{P}})$ be an abstract transition candidate. The conditions under which $(s_{\mathcal{P}}, l, s'_{\mathcal{P}}) \in \mathcal{T}^\pi_{\mathcal{P}}$ can in principle be encoded into SMT or MILP. However, as our empirical results show (Section 5), using such generic encodings and solvers severely limits scalability. Instead, here we show how to leverage Veritas which is specialized to tree ensemles.

4.1 Transition Test Without Applicability Filter

We start with the policy $\pi_{\mathcal{C}}$. Representing the test $(s_{\mathcal{P}}, l, s'_{\mathcal{P}}) \in \mathcal{T}^{\pi_{\mathcal{C}}}_{\mathcal{P}}$ as a query on $\mathcal{C}$ similar to Section 3 poses the challenge to deal with the dynamics of the transition system, involving two distinct abstract states (start and target) instead of a single input region as in the static settings for which Veritas was developed. We formulate the constraints on the target state over a *primed* copy of the state variables, as is commonly done in many settings. We handle these primed variables in Veritas queries by (temporarily) adding them as additional input variables to the tree ensembles.

Specifically, to determine whether $(s_{\mathcal{P}}, l, s'_{\mathcal{P}}) \in \mathcal{T}^{\pi_{\mathcal{C}}}_{\mathcal{P}}$, we process each l-labeled operator (g, l, u) in turn, creating an individual query $\exists s, s' \colon s, s' \models \Gamma \wedge g \wedge u \wedge s_{\mathcal{P}} \wedge s'_{\mathcal{P}}$ so that $\mathcal{C}(s) = l$ (thus, $\pi_{\mathcal{C}}(s) = l$), using the original or primed state variables within these constraints as appropriate. The box constraints Γ represent the state variable bounds and are the same in all queries. The g constraint forces s to satisfy the operator's guard. u ties the variable values in s and s' according to the operator's update. Finally, $s_{\mathcal{P}}$ and $s'_{\mathcal{P}}$ enforce that s and s' satisfy the same predicates as the respective abstract state. Note that all these constraints are either linear constraints themselves, or Boolean combinations thereof. To solve this query with Veritas, we follow the steps detailed in Section 3 to translate it into an equivalent query that uses only box constraints. The transition $(s_{\mathcal{P}}, l, s'_{\mathcal{P}})$ exists iff Veritas finds a solution to that query.

4.2 Transition Test With Applicability Filter

Casting the transition test $(s_{\mathcal{P}}, l, s'_{\mathcal{P}}) \in \mathcal{T}^{\pi^{\text{app}}_{\mathcal{C}}}_{\mathcal{P}}$ as a tree-ensemble query raises the additional complication that the applicable actions are dependent on the concrete state $s \in [s_{\mathcal{P}}]$, not known at query generation time. In order to take into account the actions' guards, we follow the principles from Section 3, augmenting the tree ensembles $\boldsymbol{T}_{l'}$ for every label l' by trees $\boldsymbol{T}^{\text{app}}_{l'}$ penalizing l' if not applicable for the given input s. Specifically, let $l' \in \mathcal{L}$ be any action label. l' is applicable in a state s iff s satisfies the disjunction $\Psi_{l'} := \bigvee_{(g,l',u) \in \mathcal{O}} g$ of guards of all l-labeled operators. Let $\boldsymbol{T}_{\Psi_{l'}}$ be the tree ensemble compilation of $\Psi_{l'}$, as described in Section 3, i.e., the tree ensemble that evaluates to $\Psi_{l'}(s) \leq \delta$ if $s \not\models \Psi_{l'}$, for δ defined in Eq. (2), and to 0 otherwise.

We make two modifications to $\mathcal{C}$: (1) we replace $\boldsymbol{T}_{l'}$ by $\boldsymbol{T}_{l'} + \boldsymbol{T}_{\Psi_{l'}}$ for every action label $l' \in \mathcal{L}, l' \neq l$ (l is handled separately); and (2) we add to $\mathcal{C}$ an auxiliary "noop" label class l_{noop}, which we assume to be applicable in all states and whose execution does not change the state, and we associate it with the tree ensemble $\boldsymbol{T}_{l_{\text{noop}}}(s) = \nu^{\boldsymbol{T}}_{\min}$ (cf.

Section 3.1). Condition (1) makes sure that the output of the modified classifier for a state s, i.e., $l_s := argmax_{l' \in \mathcal{L}}(\boldsymbol{T}_{l'} + \boldsymbol{T}_{\Psi_{l'}})(s)$, is guaranteed to be applicable in s, i.e., $s \models \Psi_{l_s}$, therewith obtaining $\pi^{\text{app}}_{\mathcal{C}}(s) = argmax_{l' \in \mathcal{L}(s)}\boldsymbol{T}_{l'}(s) = argmax_{l' \in \mathcal{L}}(\boldsymbol{T}_{l'} + \boldsymbol{T}_{\Psi_{l'}})(s) = \mathcal{C}'(s)$. The additional label class (2) is needed to handle the special case $(s_\mathcal{P}, l, s'_\mathcal{P})$, where (a) $s_\mathcal{P}$ has some l-labeled transition, but no l-labeled transition to $s'_\mathcal{P}$, while (b) $s_\mathcal{P}$ does not have an l'-labeled transition for any $l' \neq l$. Without l_{noop}, the penalties would cancel each other out, possibly creating a solution to the tree-ensemble query although $(s_\mathcal{P}, l, s'_\mathcal{P}) \notin \mathcal{T}_\mathcal{P}^{\pi^{\text{app}}_{\mathcal{C}}}$. Here, l_{noop} acts as the fallback, given that $\boldsymbol{T}_{l_{\text{noop}}}$ always returns a value higher than the penalty. Note that l_{noop} can actually only influence the result in that particular case given the choice of the value $\nu^{\boldsymbol{T}}_{\min}$.

To test whether $(s_\mathcal{P}, l, s'_\mathcal{P}) \in \mathcal{T}_\mathcal{P}^{\pi^{\text{app}}_{\mathcal{C}}}$, we proceed as described in Section 4.1: we iterate over each l-labeled operator (g, l, u), using Veritas to solve the query $\exists s, s' \colon s, s' \models \Gamma \wedge g \wedge u \wedge s_\mathcal{P} \wedge s'_\mathcal{P}$ so that $\mathcal{C}'(s) = l$ for the modified classifier $\mathcal{C}'$ given above.

5 Experiments

We ran experiments on a substantial extension of VEA's benchmarks, comparing (1) Veritas vs. Z3 and Gurobi in PPA, (2) policy quality and verifiability of tree-ensembles vs. their neural teachers, and (3) PPA vs. NUXMV. In what follows, we explain these comparisons, describe our benchmarks, and discuss our results.

Our implementation of PPA for tree-ensemble policies extends VEA's C++ code for neural policies.[1] All experiments were run on Intel Xeon E5-2650v3 CPUs, using time and memory cutoffs of 12h and 4GB. Throughout, we consider policy verification with and without applicablility filtering.

5.1 Comparison to Competing Methods

Comparison (1): Our key contribution are methods allowing to use the dedicated solver Veritas within PPA for tree-ensemble policies. Yet the necessary transition tests can also be encoded into generic constraint languages, and dispatched with generic solvers. To evaluate this alternative, we compared to SMT encodings solved with **Z3** [14], as well as to mixed-integer linear program encodings [27], solved with **Gurobi** [23]. The results are decisive: *PPA verification with Gurobi is competitive in our smallest Blocksworld benchmarks; in all other comparisons, Veritas dominates by at least an order of magnitude*. This goes to show the advantage of using a model-specific solver rather than generic ones (consistently with prior results for Veritas on classifier robustness verification [15]). In what follows, we show data only for PPA with Veritas.

Comparison (2): Our tree-ensemble policies are learned from neural teachers as detailed below. So what are the merits of either policy representation, in terms of policy quality vs. policy verifiability? To answer this question, we compare these measures across policy representations.

Comparison (3): Finally, while ours is the first work dedicated to verification of tree-ensemble policies, one can encode such policies into standard verification languages, and thus apply standard verification methods. To evaluate how our proposed PPA machinery fares against such methods, we compare to a broad range of state-of-the-art verification methods implemented in NUXMV [10]. Specifically, we experiment with the following NUXMV configurations: **bounded model checking (BMC)** [5] and **simple bounded model checking (SBMC)** [6], each using SMT instead of SAT; **explicit predicate abstraction (EPA)** within a CEGAR loop, the closest relative of the PPA approach we propose here; **implicit predicate abstraction (IPA)** [39], which runs BMC with k-induction [32] within a CEGAR loop; an SMT-based **cone of influence (COI)** algorithm [13]; and NUXMV's **IC3** implementation [12].

[1] https://zenodo.org/records/12807957

5.2 Benchmarks: Models

A benchmark for policy verification is a pair of a transition system, modeled in the JANI language [8], and a policy. We discuss the former here, and the latter in the next subsection. We consider two types of JANI models:

BInt: Bounded-Integer State Variables. We use the JANI models Blocksworld and Transport introduced by VEA [44]. We do not consider their 8-puzzle benchmark, where VEA's neural policies perform very poorly (never reach the goal), and we too failed to obtain any reasonable-performing neural or tree-ensemble policy.

Blocksworld comes in two versions, "cost-aware (CA)" and "cost-ignore (CI)" that distinguish whether or not the policy receives as input a subset of state variables encoding action cost. For Transport, we introduce a new domain version which includes an additional *feature*, namely a state variable that tracks the number of packages to the left of the truck's position. This feature is beneficial for training safe policies, for both types of policy representations. We also introduce a new BInt benchmark, Beluga, based on a logistics application at Airbus. Recent work [37] introduced Beluga as a PDDL benchmark; here we adapt it to the JANI and policy verification context. Beluga is a factory logistics problem where product parts arriving on a Beluga transport plane must be sent to a factory in a particular order, potentially different from the order in which they are arriving. To compensate for a possible mismatch, there is a number of racks in which arriving parts can be stored temporarily. The start condition permits arbitrary orderings of the arriving parts. The goal requires having sent all parts to the factory, under non-determinism which part will be requested next at each step. A state is unsafe if all racks are occupied (modelling the practical guideline to keep at least one rack in store for any parts arriving out of schedule).

BReal: Real-Valued State Variables. Policy verification in Jani models with real-valued state variables has so far not been considered. We contribute a new benchmark following previous work by Amir et al. [1] on *Robotics Turtlebot 3*, a robot widely used in robotics research [2, 1]. The robot navigates in a continuous space, trying to reach a target position while avoiding obstacles. The sensing consists of multiple continuous (limited-range) lidar sensors estimating the distance to obstacles. The robot can perform three actions: rotate left, rotate right and move forward. Amir et al. train neural policies to control the robot. They perform a limited form of safety verification by encoding *single-step collision avoidance* into SMT, deciding whether there exist states where the robot is directly in front of an obstacle yet the policy decides to move forward.

Here, we extend this to *general collision avoidance*, deciding whether the robot ever moves into an obstacle. We scale this verification benchmark by a start condition fixing a distance of K steps to the nearest obstacle.

We model the Turtlebot transition semantics in JANI. The physical environment of Turtlebot however involves non-linear sensor updates. Our JANI model over-approximates these via non-deterministic action outcomes. Safety in this over-approximation implies safety in the actual environment; while unsafe paths might be

Table 1. Results on BInt benchmarks for neural-network (NN; Xh hidden layer size X) and tree-ensemble (GB, RF) policies with and without applicable actions filtering. Fid: fidelity, GFrac: fraction of goal-reaching simulations, Rew: Simulation runs' average reward, Safe: Verfication outcome. PPA, BMC, SBMC, COI, IPA, IC3: total verification runtime (sec). EPA runs out of memory everywhere and is hence omitted.

			App Filter Enabled									No App Filter								
							NuXMV									NuXMV				
Jani Model	Policy	Fid	GFrac	Rew	Safe?	PPA	BMC	SMBC	COI	IPA	IC3	GFrac	Rew	Safe?	PPA	BMC	SBMC	COI	IPA	IC3
	NN (16h)		1.00	172	Yes	14	—	—	—	—	—	1.00	172	Yes	3	—	—	—	—	—
	GB	1.00	1.00	172	Yes	4	—	—	—	134	1683	1.00	172	Yes	0	—	—	—	160	1020
	RF	1.00	1.00	172	Yes	4	—	—	—	95	391	1.00	172	Yes	1	—	—	—	35	420
	NN (32 h)		1.00	172	Yes	70	—	—	—	—	—	1.00	172	Yes	20	—	—	—	—	—
4Blocks CI	GB	1.00	1.00	172	Yes	4	—	—	12	4	3	1.00	172	Yes	0	—	—	11	4	6
	RF	1.00	1.00	172	Yes	3	—	—	15	0	42	1.00	172	Yes	0	—	—	9	1	11
	NN (64 h)		1.00	172	Yes	40768	—	—	—	—	—	1.00	173	Yes	2384	—	—	—	—	—
	GB	1.00	1.00	172	Yes	8	—	—	—	2594	2041	1.00	172	Yes	2	—	—	—	77	1030
	RF	1.00	1.00	172	Yes	4	—	—	—	—	34749	1.00	172	Yes	1	—	—	—	1211	3757
	NN (64 h)		0.97	156	Yes	6030	—	—	—	—	—	0.98	156	Yes	2154	—	—	—	—	—
6Blocks CI	GB	0.99	0.99	157	Yes	2595	—	—	—	—	—	0.99	156	Yes	30	—	—	—	—	—
	RF	1.00	0.99	157	Yes	638	—	—	—	—	—	0.99	156	Yes	9	—	—	—	—	—
	NN (64 h)		0.60	82	—	—	—	—	—	—	—	0.60	73	Yes	12717	—	—	—	—	—
8Blocks CI	GB	0.98	0.65	90	—	—	—	—	—	—	—	0.65	84	Yes	6393	—	—	—	—	—
	RF	0.99	0.70	98	—	—	—	—	—	—	—	0.70	92	—	—	—	—	—	—	—
	NN (32 h)		1.00	172	Yes	2251	—	—	—	—	—	1.00	172	Yes	197	—	—	—	—	—
	GB	1.00	1.00	172	Yes	5	—	—	—	—	—	1.00	172	Yes	7	—	—	—	—	—
4Blocks CA	RF	0.99	0.00	2	Yes	5	—	—	1332	237	551	0.00	-26	Yes	0	—	—	1891	161	578
	NN (64 h)		1.00	172	—	—	—	—	—	—	—	1.00	173	—	—	—	—	—	—	—
	GB	1.00	1.00	172	Yes	6	—	—	—	—	—	1.00	172	Yes	2	—	—	—	—	—
	RF	1.00	1.00	172	Yes	4	—	—	—	—	—	1.00	172	Yes	1	—	—	—	—	—
	NN (64 h)		0.94	150	—	—	—	—	—	—	—	0.95	150	Yes	16853	—	—	—	—	—
6Blocks CA	GB	0.99	0.96	151	Yes	19965	—	—	—	—	—	0.96	151	Yes	1600	—	—	—	—	—
	RF	0.99	0.97	153	Yes	9343	—	—	—	—	—	0.97	153	Yes	1135	—	—	—	—	—
	NN (64 h)		0.00	-8	—	—	—	—	—	—	—	0.00	-37	—	—	—	—	—	—	—
8Blocks CA	GB	0.76	0.00	-12	—	—	—	—	—	—	—	0.00	-30	—	—	—	—	—	—	—
	RF	0.35	0.00	-13	—	—	—	—	—	—	—	0.00	-50	—	—	—	—	—	—	—
	NN (16h)		0.99	105	No	8927	774	1377	—	2623	—	0.98	102	No	17652	—	—	—	35	—
Transport	GB	0.99	0.91	93	No	887	14614	7089	—	—	—	0.77	67	No	—	5116	2544	18185	—	—
	RF	0.99	0.91	99	No	97	—	—	—	—	—	0.77	71	Yes	188	—	—	—	—	—
	NN (16h)		0.98	149	Yes	2521	—	—	—	—	—	0.03	-83	Yes	373	—	—	—	—	—
	GB	0.86	1.00	142	Yes	21	—	—	5	0	2	0.01	-84	Yes	6	—	134	5	0	3
Transport +	RF	0.79	1.00	119	Yes	86	—	5617	5	0	10	0.00	-85	Yes	66	—	288	2	0	19
Feature	NN (64h)		0.97	148	—	—	—	—	—	—	—	0.01	-92	—	—	—	—	—	—	—
	GB	0.94	1.00	151	Yes	35	—	—	2517	138	263	0.00	-93	Yes	40	—	23955	2149	143	172
	RF	0.97	1.00	153	Yes	22	—	—	1658	67	327	0.01	-93	Yes	6	—	—	—	80	447
	NN (64h)		0.90	134	—	—	—	—	—	—	—	0.00	-100	Yes	4	—	—	—	—	—
Beluga4Parts	GB	0.99	1.00	158	—	—	—	0	0	24	542	0.00	-100	Yes	1	—	0	0	23	157
	RF	0.98	1.00	157	Yes	2	—	2	2	52	2047	0.00	-100	Yes	1	—	2	1	43	165
	NN (256h)		0.99	161	—	—	—	—	—	—	—	0.00	-100	—	—	—	—	—	—	—
Beluga5Parts	GB	0.99	0.85	119	—	—	—	3	3	7	27	0.00	-100	Yes	0	—	3	3	5	15
	RF	0.79	0.46	62	Yes	16	—	12	14	—	—	0.00	-100	Yes	8	—	11	16	—	—
	NN (64h)		0.84	114	—	—	—	—	—	—	—	0.00	-100	—	—	—	—	—	—	—
Beluga6Parts	GB	0.89	0.60	73	—	—	—	2	2	—	—	0.00	-100	Yes	11	—	1	2	—	—
	RF	0.92	0.01	-22	—	—	—	—	10	—	—	0.00	-100	Yes	14	—	—	13	—	—

spurious (only exist in the over-approximation).

We remark that, by extending VEA's benchmark set in this manner, as a side effect of our work we also obtain new results for PPA verifcation of neural policies, in particular the first results for such verification in an infinite state space (Turtlebot). We briefly discuss these additional results below.

5.3 *Benchmarks: Tree-Ensemble Policies*

We train tree-ensemble policies through imitation learning, using for each benchmark a neural-network policy as the teacher (teacher selection is described below). For training the policies, we consider gradient-boosted trees (short GB) [11] and random forests (short RF) [7], with varying hyperparameter settings: tree depth in $\{4, 6, 8, 10, 15\}$; number of trees in $\{5, 10, 20, 30\}$; and learning rate in $\{0.4, 0.6, 0.8\}$. The training-set generation and policy selection procedures differ between BInt and BReal.

BInt. The training set is generated by executing the teacher policy in the Jani model from 5000 randomly sampled start states, with the applicability filter enabled, collecting all pairs of state and action value predictions on these runs. We obtain different tree-ensemble policies from this training set by fitting a multi-target regression model [3] for each hyperparameter combination. To choose among the trained policies, we evaluated them through simulation runs from 10,000 randomly chosen start states (the test set; same for all policies) with the applicability filter enabled, considering three metrics: **Reward**, the reward function used for neural policy training; **GFrac**, the fraction of simulation runs where the policy reached a goal state; and **Fidelity**, the fraction of states for which the tree-ensemble policy selects the same action as its teacher. We select the GB and the RF tree ensemble that achieve the highest reward, discarding policies that were already found unsafe during simulation. If several policies obtain the highest reward, we break ties by GFrac, depth, and then number of trees, thus encouraging selecting smaller tree ensembles.

BReal. The overapproximating Turtlebot Jani model doesn't support simulation runs. Instead, for the training set, we randomly generated 1,000,000 environment states. Also, we used a classification loss here, which works well on the only 3 actions in this domain. Otherwise, training proceeds as above. To choose among the trained policies, we sampled 10,000 environment states (same for all policies), and selected the GB respectively RF policies achieving highest fidelity on this test set. Similarly to before, we break ties by depth, and then number of trees.

Teacher Policies. For each JANI model, like VEA we consider neural policies with different hidden layer sizes. To obtain high-quality tree-ensemble policies, we chose as teachers only those neural policies whose average reward measured on the test set (cf. above) is within 1% of the best-performing neural policy. Also, we discard neural policies found unsafe during the test runs.

In Blocksworld, we use VEA's neural policies. For Transport+Feature and Beluga, we trained new policies ourselves using Q-learning. In Transport as already used by VEA, VEA's policies never reach the goal, and we were not able to learn a useful neural policy with Q-learning either. We instead designed a handmade domain-specific policy, which we used as a teacher for the neural policy in imitation learning. The neural policy was then used as teacher for the tree-ensembles as elsewhere. For Turtlebot, we select two policies from the ones trained by Amir et al. [1], one proved safe and one proved unsafe for single-step collision avoidance.

5.4 *Results*

As previously discussed, the comparison (1) of Veritas against Z3 and Gurobu was decisive so data for Z3 and Gurobi is not included here. In what follows, we discuss the comparison (2) between PPA verification of neural vs. tree-ensemble policies, as well as the comparison (3) between PPA and state-of-the-art methods implemented in NUXMV. We organize the discussion into one section for the BInt, and one for the BReal benchmarks.

BInt. Table 1 shows our results for the BInt benchmarks. Let us first discuss comparison (3). Explicit predicate abstraction (EPA) is not in Table 1 as it runs out of memory everywhere. The same is the case in Turtlebot discussed below. Hence our PPA approach outclasses its closest relative in NUXMV.

With respect to the other NUXMV configurations, for tree-ensemble policies PPA is highly complementary. It outperforms NUXMV in the Blocksworld, and mostly also in Transport. In Transport+Feature and Beluga, without applicability filtering PPA and NUXMV are roughly on par, with PPA exhibiting consistently good performance while the best NUXMV configuration depends on the domain. With applicability filtering, there are NUXMV configurations that outperform PPA.

For neural policies, the picture of PPA vs. NUXMV is consistent with VEA's findings [45]: NUXMV does not succeed in anything except finding unsafe paths in Transport.

Consider now comparison (2). In terms of policy quality, our tree-ensemble policies typically closely replicate their neural teachers. In almost all cases, fidelity is close to 1.00, and goal fraction as well as reward match that of the neural teacher. The only exception is Beluga, where it is sometimes challenging to learn high-quality policies. In all domains, to the extent we can verify it, the best tree-ensemble policy is safe whenever the neural teacher policy is.

In terms of policy verifiability, the tree-ensemble policies are consistently verified faster than their neural teachers, often by orders of magnitude (the only exception is Transport 16h GB without applicability filter). There are several instances where the tree-ensemble policies are successfully verified to be safe, sometimes within seconds, while the verification of their teachers times out after 12 hours.

Table 2. Results for Turtlebot. Abbreviations as in Table 1. Neural policies trained by Amir et al. [1], one safe (top), one unsafe (bottom).

Jani model	Policy	Fid	Safe?	PPA
1 step-Turtlebot	NN (16h)	—	Yes	3.9
	GB	0.98	Yes	629.3
	RF	0.99	Yes	2.9
8 step-Turtlebot	NN (16h)	—	Yes	3.8
	GB	0.98	Yes	633.4
	RF	0.99	Yes	2.9
14 step-Turtlebot	NN (16h)	—	Yes	3.9
	GB	0.96	Yes	630.9
	RF	0.99	Yes	2.9
1 step-Turtlebot	NN (16h)	—	No	0.9
	GB	0.98	Yes	1236.5
	RF	0.93	Yes	1.2
8 step-Turtlebot	NN (16h)	—	No	80.3
	GB	0.98	Yes	69.3
	RF	0.93	Yes	1.2
14 step-Turtlebot	NN (16h)	—	No	921.3
	GB	0.98	Yes	597.2
	RF	0.93	Yes	1.2

BReal. Table 2 shows the results for Turtlebot. All actions are always applicable here so we do not distinguish with vs. without applicability filtering. Instead, we distinguish two neural polices trained by Amir et al. [1], a safe one (top) vs. an unsafe one (bottom).

Comparison (3) is quickly done in Turtlebot: all NUXMV configurations run out of time or memory everywhere and are thus not included in the table.

Regarding comparison (2), for the safe neural policy the most remarkable observation is the constant scaling in K. PPA always identifies the small reason for safety despite the growing obstacle distance. For the unsafe neural policy, verification difficulty scales with K for the neural and GB policies, while the RF policies are verified quickly. Oddly, the tree policies are safe here (in the last step, unlike their neural teacher, they choose to rotate instead of driving into the wall). This is a coincidence in the learning process, there is no incentive to behave differently from the teacher.

6 Conclusion

We show how to use Veritas in PPA, establishing the first methodology dedicated to verification of tree-ensemble policies. We find that (1) using Veritas is vastly superior to using generic solvers; (2) our tree ensembles typically have similar quality as their neural teachers, while being much faster to verify; and (3) PPA is complementary to state-of-the-art methods in NUXMV, outperforming them in half of our benchmarks.

An obvious opportunity for future work is to tailor some of the competitive algrithms in NUXMV to ML models, in particular to tree ensembles. Regarding tree ensembles as a policy representation, an important topic is automatic feature generation (e.g. [16]), and extending PPA verification to deal with such features. A very general hypothesis is that, *within the realm of problems where a full verification can be hoped for, tree ensembles often offer a better trade-off between policy quality vs. verifiability than neural networks*. Much more work is needed to explore this hypothesis in full.

Acknowledgements

This work was funded by the European Union's Horizon Europe Research and Innovation program under the grant agreement TUPLES No 101070149.

References

[1] G. Amir, D. Corsi, R. Yerushalmi, L. Marzari, D. Harel, A. Farinelli, and G. Katz. Verifying learning-based robotic navigation systems. In *Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part I*, volume 13993 of *Lecture Notes in Computer Science*. Springer, 2023.

[2] R. Amsters and P. Slaets. Turtlebot 3 as a robotics education platform. In M. Merdan, W. Lepuschitz, G. Koppensteiner, R. Balogh, and D. Obdržálek, editors, *Robotics in Education - Current Research and Innovations, 10th RiE, Vienna, Austria, April 10-12, 2019*, volume 1023 of *Advances in Intelligent Systems and Computing*. Springer, 2020.

[3] J. Ba and R. Caruana. Do deep nets really need to be deep? In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[4] C. W. Barrett and C. Tinelli. Satisfiability modulo theories. In *Handbook of Model Checking*. Springer, 2018.

[5] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without bdds. 1999.

[6] A. Biere, K. Heljanko, T. Junttila, T. Latvala, and V. Schuppan. Linear encodings of bounded ltl model checking. *lmcs*, Volume 2, Issue 5, Nov. 2006. doi: 10.2168/LMCS-2(5:5)2006.

[7] L. Breiman. Random forests. *Machine learning*, 45, 2001.

[8] C. E. Budde, C. Dehnert, E. M. Hahn, A. Hartmanns, S. Junges, and A. Turrini. JANI: quantitative model and tool interaction. In *23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'17)*, 2017.

[9] L. Cascioli, L. Devos, and J. Davis. Multi-class robustness verification for tree ensembles. In *Verifying Learning AI Systems Workshop*, 2023.

[10] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta. The nuxmv symbolic model checker. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*. Springer, 2014.

[11] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[12] A. Cimatti and A. Griggio. Software model checking via ic3. 2012.

[13] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2001.

[14] L. de Moura and N. Bjorner. Z3: An Efficient SMT Solver. In C. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2008*. Springer, 2008.

[15] L. Devos, W. Meert, and J. Davis. Versatile verification of tree ensembles. In *38th International Conference on Machine Learning*, 2021.

[16] D. Drexler and J. Seipp. DLPlan: Description logics state features for planning. In *ICAPS 2023 System Demonstrations and Exhibits*, 2023.

[17] S. Dutta, X. Chen, and S. Sankaranarayanan. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *22nd International Conference on Hybrid Systems: Computation and Control (HSCC'19)*, 2019.

[18] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 2001.

[19] S. Garg, A. Bajpai, et al. Size independent neural transfer for RDDL planning. In *ICAPS*, 2019.

[20] S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In O. Grumberg, editor, *9th International Conference on Computer Aided Verification (CAV'97)*, Lecture Notes in Computer Science. Springer, 1997.

[21] L. Grinsztajn, E. Oyallon, and G. Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In *Advances in Neural Information Processing Systems*, volume 35, 2022.

[22] E. Groshev, M. Goldstein, A. Tamar, S. Srivastava, and P. Abbeel. Learning generalized reactive policies using deep neural networks. In *ICAPS*, 2018.

[23] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL https://www.gurobi.com.

[24] S. Huang, J. F. andf Wenchao Li, X. Chen, and Q. Zhu. Reachnn: Reachability analysis of neural-network controlled systems. *ACM Transactions on Embedded Computing Systems*, 18, 2019.

[25] M. Issakkimuthu, A. Fern, and P. Tadepalli. Training deep reactive policies for probabilistic planning problems. In *ICAPS*, 2018.

[26] R. Ivanov, T. J. Carpenter, J. Weimer, R. Alur, G. J. Pappas, and I. Lee. Verifying the safety of autonomous systems with neural network controllers. *ACM Transactions on Embedded Computing Systems*, 20(1), 2021.

[27] A. Kantchelian, J. D. Tygar, and A. D. Joseph. Evasion and hardening of tree ensemble classifiers. In *International Conference on Machine Learning*, 2015.

[28] R. Karia and S. Srivastava. Learning generalized relational heuristic networks for model-agnostic planning. In *AAAI*, 2021.

[29] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljic, D. L. Dill, M. Kochenderfer, and C. Barrett. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In I. Dillig and S. Tasiran, editors, *Computer Aided Verification. CAV 2019*, LNCS 11561, Cham, 2019. Springer. https://doi.org/10.1007/978-3-030-25540-4_26.

[30] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540), 2015.

[31] G. Pagallo. Learning dnf by decision trees. In N. S. Sridharan, editor, *11th International Joint Conference on Artificial Intelligence (IJCAI'89)*, Detroit, MI, Aug. 1989. Morgan Kaufmann.

[32] M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a sat-solver. In *Formal Methods in Computer-Aided Design, Third International Conference, FMCAD 2000, Austin, Texas, USA, November 1-3, 2000, Proceedings*, volume 1954 of *Lecture Notes in Computer Science*. Springer, 2000.

[33] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529, 2016.

[34] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. A general reinforcement learning algorithm that masters Chess, Shogi, and Go through self-play. *Science*, 362 (6419), 2018.

[35] S. Ståhlberg, B. Bonet, and H. Geffner. Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. In *ICAPS 2022*.

[36] S. Ståhlberg, B. Bonet, and H. Geffner. Learning generalized policies without supervision using gnns. In *19th International Conference on Principles of Knowledge Representation and Reasoning (KR'22)*, 2022.

[37] M. Steinmetz, Thiébaux, D. Höller, and F. Teichteil-Königsbuch. Explaining the space of SSP policies via policy-property dependencies: Complexity, algorithms, and relation to multi-objective planning. In *ICAPS*, 2024.

[38] X. Sun, H. Khedr, and Y. Shoukry. Formal verification of neural network controlled autonomous systems. In *International Conference on Hybrid Systems: Computation and Control (HSCC'19)*, 2019.

[39] S. Tonetta. Abstract model checking without computing the abstraction. 2009.

[40] S. Toyer, S. Thiébaux, F. W. Trevizan, and L. Xie. Asnets: Deep learning for generalised planning. *Journal of Artificial Intelligence Research*, 2020.

[41] H. Tran, F. Cai, D. M. Lopez, P. Musau, T. T. Johnson, and X. D. Koutsoukos. Safety verification of cyber-physical systems with reinforcement learning control. *ACM Transactions on Embedded Computing Systems*, 18(5s), 2019.

[42] G. S. Tseitin. On the complexity of derivation in propositional calculus. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970*. Springer, Berlin, Heidelberg, 1983.

[43] M. Vinzent and J. Hoffmann. Neural policy safety verification via predicate abstraction: Applicability filtering. In *ICAPS*, 2024.

[44] M. Vinzent, M. Steinmetz, and J. Hoffmann. Neural network action policy verification via predicate abstraction. In *ICAPS*, 2022.

[45] M. Vinzent, S. Sharma, and J. Hoffmann. Neural policy safety verification via predicate abstraction: CEGAR. In *AAAI*, 2023.