# Policy-Specific Abstraction Predicate Selection in Neural Policy Safety Verification

**Marcel Vinzent**[1], **Min Wu**[2], **Haoze Wu**[2], **Jörg Hoffmann**[1,3]

[1] Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
[2] Department of Computer Science, Stanford University, Stanford, United States
[3] German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany
{vinzent, hoffmann}@cs.uni-saarland.de, {minwu, haozewu}@cs.stanford.edu

## Abstract

Neural networks (NN) are an increasingly important representation of action policies $\pi$. Recent work has introduced *policy predicate abstraction (PPA)* to prove safety of such $\pi$, through over-approximating the state space subgraph induced by $\pi$. Counterexample-guided abstraction refinement (CEGAR) drives the verification process, iteratively building the PPA for the current set of predicates, finding an abstract unsafe path $\sigma_{\mathcal{P}}$, and selecting new predicates based on sources of spuriousness in $\sigma_{\mathcal{P}}$. Compared to standard predicate abstraction, this involves a new source of spuriousness, where the necessary concrete transition exists but the neural network $\pi$ selects a different action. Prior work tackles this source of spuriousness naïvely, based on the concrete states involved. Here we introduce a method that instead selects new abstraction predicates specific to policy behavior, trying to characterize $\pi$'s decision boundary. Inspired by ideas from formal explainable AI (XAI) in computer vision, we use SMT reasoning to identify a state variable set $\mathcal{V}_{rel}$ relevant to $\pi$, in the sense that $\pi$'s decision is invariant against perturbation of the irrelevant variables $\mathcal{V} \setminus \mathcal{V}_{rel}$. We then add new predicates based on $v \in \mathcal{V}_{rel}$. We introduce and empirically evaluate a number of variants of this approach. The results show that the approach can sometimes improve CEGAR performance, though there is no clear winning configuration.

## 1 Introduction

Neural networks (NN) are an increasingly important representation of action policies in many contexts, including AI planning (Issakkimuthu, Fern, and Tadepalli 2018; Groshev et al. 2018; Garg, Bajpai, and Mausam 2019; Toyer et al. 2020; Stahlberg, Bonet, and Geffner 2022). But how to verify that such a policy is safe? Given a policy $\pi$, a **start condition** $\phi_0$, and an **unsafety condition** $\phi_u$, how to verify whether a state $s^u \models \phi_u$ is reachable from a state $s^0 \models \phi_0$ under $\pi$? Such verification is potentially very hard as it compounds the state space explosion with the difficulty of analyzing even single NN decision episodes.

Research has recently begun to address this question. A prominent line of works addresses neural controllers of dynamical systems, where the NN outputs a vector $u$ of reals forming input to a continuous state-evolution function $f$ (Sun, Khedr, and Shoukry 2019; Tran et al. 2019; Huang et al. 2019; Dutta, Chen, and Sankaranarayanan 2019). Recent work extends this thread to hybrid systems, addressing smooth (tanh/sigmoid) activation functions by compilation into such systems (Ivanov et al. 2021). Another thread explores bounded-length verification of neural controllers (Akintunde et al. 2018, 2019; Amir, Schapira, and Katz 2021).

Here we follow up on the work by Vinzent et al. (2022; 2023) (henceforth: **Vea**), which tackles NN policies $\pi$ with ReLU activation functions taking discrete action choices in non-deterministic state spaces over bounded-integer state variables $\mathcal{V}$. The approach extends **predicate abstraction (PA)** (Graf and Saïdi 1997; Ball et al. 2001; Henzinger et al. 2004) to what Vea baptize **policy predicate abstraction (PPA)**. Like PA, PPA builds an over-approximating abstraction defined through a set $\mathcal{P}$ of **predicates**, where each $p \in \mathcal{P}$ is a linear constraint over the state variables (e.g. $x = 7$ or $x \leq y$). Unlike PA however, PPA abstracts not the full state space $\Theta$, but the **policy-restricted** state space $\Theta^\pi$, i.e., the state-space subgraph containing only the transitions taken by $\pi$. Vea's method builds the fragment of $\Theta^\pi_{\mathcal{P}}$ – the predicate abstraction of $\Theta^\pi$ – reachable from $\phi_0$, and checks whether $\phi_u$ is reached. If this is not the case, then $\pi$ is safe.

To find a suitable predicate set $\mathcal{P}$ automatically, Vea employ counterexample-guided abstraction refinement (CEGAR) (Clarke et al. 2003), iteratively building the PPA for the current set of predicates, finding an abstract unsafe path $\sigma_{\mathcal{P}}$, and selecting new predicates based on sources of spuriousness in $\sigma_{\mathcal{P}}$ (if no $\sigma_{\mathcal{P}}$ exists then $\pi$ is proved safe, if $\sigma_{\mathcal{P}}$ is not spurious then $\pi$ is proved unsafe). Vea show, on a collection of benchmarks adapted from AI planning, that this approach tends to outperform encodings into the state-of-the-art verification tool NUXMV (Cavada et al. 2014), in particular being the only approach that succeeds in proving policies safe.

In the present paper, we hone in on the sources of spuriousness in the abstract unsafe path $\sigma_{\mathcal{P}}$, and the adequate selection of new predicates to address these sources. This is an important and subtle problem because, whereas in standard PA spurious behavior is always due to the over-approximation in abstract state transitions, in PPA the reason may instead be a decision of the neural network $\pi$. To see this, consider a concrete state $s$ in an abstract state $s_{\mathcal{P}}$, and say that the abstract path next proceeds to $s'_{\mathcal{P}}$ via label $l$. In PA, this is possible iff there is a concrete transition $(s, l, s')$ s.t. $s' \in s'_{\mathcal{P}}$. In PPA, we additionally need that $\pi(s) = l$.

If this is not the case, then the needed transition behavior is there, but the needed NN decision is not.

This source of spuriousness calls for *new predicates specific to policy behavior, distinguishing states within $s_\mathcal{P}$ where $\pi$ selects vs. does not select $l$*, and therewith characterizing the relevant decision boundary of the neural network $\pi$. Vea did not tackle this challenge. Instead, they opted for a naïve method, based on a **witness state** $s_w \in s_\mathcal{P}$ (found by PPA) for the abstract transition $(s_\mathcal{P}, l, s'_\mathcal{P})$. They introduce new predicates allowing to distinguish $s$ from $s_w$. This disregards the question which aspects of $s_w$ are actually relevant to $\pi$'s decision.

Here we tackle this question in detail. Inspired by ideas from formal explainable AI (XAI) in computer vision (Wu, Wu, and Barrett 2022), we say that a set of state variables $\mathcal{V}_{irr} \subseteq \mathcal{V}$ is **$\epsilon$-irrelevant** if the decision of $\pi$ in $s_w$ is invariant against perturbation of $\mathcal{V}_{irr}$ by at most $\epsilon$ while keeping all other variables fixed. If $\mathcal{V}_{irr}$ is maximal, we say $\mathcal{V}_{rel} = \mathcal{V} \setminus \mathcal{V}_{irr}$ is **$\epsilon$-relevant**. We use SMT reasoning specialized to NN (Katz et al. 2017, 2019) to compute an $\epsilon$-relevant set $\mathcal{V}_{rel}$ in a loop over $v \in \mathcal{V}$. We get rid of the algorithm parameter $\epsilon$ through an outer loop using binary search to identify a minimal $\epsilon$ such that an $\epsilon$-relevant $\mathcal{V}_{rel}$ exists. The incurred SMT overhead is negligible compared to the SMT effort required in PPA anyhow, so that this methodology is feasible.

We also experiment with the form of new predicates introduced, in particular relying on $\epsilon$ to identify a value between $s_w$ and $s$ – intuitively, estimating the location of $\pi$'s decision boundary – to be used in the new abstraction predicate.

We evaluate our techniques on Vea's benchmarks, comparing a broad range of algorithm configurations. The results show that the approach can sometimes improve CEGAR performance, though there is no clear winning configuration.

## 2 Preliminaries

We consider the policy verification setting as introduced by Vea. A state space is a tuple $\langle \mathcal{V}, \mathcal{L}, \mathcal{O} \rangle$ of **state variables** $\mathcal{V}$, **action labels** $\mathcal{L}$, and **operators** $\mathcal{O}$. For each variable $v \in \mathcal{V}$ the domain $D_v \neq \emptyset$ is a bounded integer interval. We denote by $Exp$ the set of **linear integer expressions** over $\mathcal{V}$, of the form $d_1 \cdot v_1 + \cdots + d_r \cdot v_r + c$ with $d_1, \ldots, d_r, c \in \mathbb{Z}$. $C$ denotes the set of **linear integer constraints** over $\mathcal{V}$, i.e., constraints of the form $e_1 \bowtie e_2$ with $\bowtie \in \{\leq, =, \geq\}$ and $e_1, e_2 \in Exp$, and Boolean combinations thereof. An **operator** $o \in \mathcal{O}$ is a tuple $(g, l, u)$ with **label** $l \in \mathcal{L}$, **guard** $g \in C$, and (partial) **update** $u \colon \mathcal{V} \rightarrow Exp$.

A (partial) **variable assignment** $s$ over $\mathcal{V}$ is a function with domain $dom(s) \subseteq \mathcal{V}$ and $s(v) \in D_v$ for $v \in dom(s)$. Given $s_1, s_2$, we denote by $s_1[s_2]$ the update of $s_1$ by $s_2$, i.e., $dom(s_1[s_2]) = dom(s_1) \cup dom(s_2)$ with $s_1[s_2](v) = s_2(v)$ if $v \in dom(s_2)$, else $s_1[s_2](v) = s_1(v)$. By $e(s)$ we denote the evaluation of $e \in Exp$ over $s$, and by $\phi(s)$ the evaluation of $\phi \in C$. If $\phi(s)$ evaluates to true, we write $s \models \phi$.

The **state space** of $\langle \mathcal{V}, \mathcal{L}, \mathcal{O} \rangle$ is a labeled transition system (LTS) $\Theta = \langle \mathcal{S}, \mathcal{L}, \mathcal{T} \rangle$. The set of **states** $\mathcal{S}$ is the (finite) set of all 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 $s \models g$ and $s' = s[u(s)]$, where $u(s) = \{v \mapsto u(v)(s) \mid v \in dom(u)\}$. We also write $s \models o$ for $s \models g$, and abbreviate $s[\![o]\!]$ for $s[u(s)]$.

Observe that the separation between action labels and operators allows both, state-dependent effects (different operators with the same label $l$ applicable in different states) and action outcome non-determinism (different operators with the same label $l$ applicable in the same state).

A **policy** $\pi$ is a function $\mathcal{S} \rightarrow \mathcal{L}$. We consider $\pi$ represented by a **neural network (NN)**. While Vea's approach applies, in principle, to arbitrary NN, its current implementation is restricted to feed-forward NN with **rectified linear unit (ReLU)** activation functions $ReLU(x) = \max(x, 0)$. These NN consist of an input layer, arbitrarily many hidden layers, and an output layer with one neuron per label $l$; $\pi$ is obtained by applying argmax to the output layer.

We next introduce basic safety notations pertaining to $\Theta$ as a whole; we will later modify these to consider policy safety instead. A **safety property** is a pair $\rho = (\phi_0, \phi_u)$ where $\phi_0, \phi_u \in C$. Here, $\phi_u$ identifies the set of unsafe states that should be unreachable from the set of possible start states represented by $\phi_0$. $\Theta$ is **unsafe** with respect to $\rho$ iff there exists a path $\langle s^0, l^0, \ldots, s^{n-1}, l^{n-1}, s^n \rangle$ in $\Theta$ such that $s^0 \models \phi_0$ and $s^i n \models \phi_u$. Otherwise $\Theta$ is **safe**.

Predicate abstraction (Graf and Saïdi 1997) is a well-established technique to prove safety. Assume a set of predicates $\mathcal{P} \subseteq C$. An **abstract state** $s_\mathcal{P}$ is a (complete) truth value assignment over $\mathcal{P}$. The abstraction of a (concrete) state $s \in \mathcal{S}$ is the abstract state $s|_\mathcal{P}$ with $s|_\mathcal{P}(p) = p(s)$ for each $p \in \mathcal{P}$. Conversely, $[s_\mathcal{P}] = \{s' \in \mathcal{S} \mid s'|_\mathcal{P} = s_\mathcal{P}\}$ denotes the set of all concrete states represented by $s_\mathcal{P}$. The **predicate abstraction** of $\Theta$ over $\mathcal{P}$ is then the LTS $\Theta_\mathcal{P} = \langle \mathcal{S}_\mathcal{P}, \mathcal{L}, \mathcal{T}_\mathcal{P} \rangle$, where $\mathcal{S}_\mathcal{P}$ is the set of all abstract states over $\mathcal{P}$, and $\mathcal{T}_\mathcal{P} = \{(s|_\mathcal{P}, l, s'|_\mathcal{P}) \mid (s, l, s') \in \mathcal{T}\}$. To compute $\Theta_\mathcal{P}$, one must solve what we call the **abstract transition problem** for every possible abstract transition: $(s_\mathcal{P}, l, s'_\mathcal{P}) \in \mathcal{T}_\mathcal{P}$ iff there exists an $l$-labeled operator $o \in \mathcal{O}$ and a concrete state $s_w \in [s_\mathcal{P}]$, a **transition witness**, such that $s_w \models o$ and $s_w[\![o]\!] \in [s'_\mathcal{P}]$. Such abstract transition problems are routinely encoded into satisfiability modulo theories (SMT) (Barrett and Tinelli 2018), using off-the-shelf solvers like $Z3$ (de Moura and Bjørner 2008).

Analogously to safety in $\Theta$, the abstract state space $\Theta_\mathcal{P}$ is said to be **unsafe** with respect to a safety property $\rho = (\phi_0, \phi_u)$ iff there exists a path $\sigma_\mathcal{P} = \langle s^0_\mathcal{P}, l^0, \ldots, l^{n-1}, s^n_\mathcal{P} \rangle$ in $\Theta_\mathcal{P}$ such that $s^0_\mathcal{P} \models \phi_0$ and $s^n_\mathcal{P} \models \phi_u$, where $s_\mathcal{P} \models \phi$ iff there exists $s \in [s_\mathcal{P}]$ such that $s \models \phi$. Otherwise $\Theta_\mathcal{P}$ is **safe**, in which case $\Theta$ is safe as well.

A concrete path $\langle s^0, l^0, \ldots, l^{n-1}, s^n \rangle$ such that $s^0 \models \phi_0$, $s^i \in [s^i_\mathcal{P}]$ for $i \in \{0, \ldots, n\}$ and $s^n \models \phi_u$ is a **concretization** of $\sigma_\mathcal{P}$. If there does not exist a concretization for $\sigma_\mathcal{P}$, then $\sigma_\mathcal{P}$ is **spurious**. Counterexample-guided abstraction refinement (**CEGAR**) (e.g. (Clarke et al. 2003; Henzinger et al. 2004; Gupta and Strichman 2005; Podelski and Rybalchenko 2007)) iteratively removes such spurious abstract paths by refining $\mathcal{P}$, until either the abstraction is proven safe or a non-spurious abstract path is found.

## 3 PPA & CEGAR (Vea)

We next briefly revisit Vea as the direct background for our work.

### Policy Predicate Abstraction

Safety of a given policy $\pi$ is proved via safety of the **policy-restricted state space** $\Theta^{\pi}$, the subgraph $\langle \mathcal{S}, \mathcal{L}, \mathcal{T}^{\pi} \rangle$ of $\Theta$ with $\mathcal{T}^{\pi} = \{(s, l, s') \in \mathcal{T} \mid \pi(s) = l\}$. In turn, safety of $\Theta^{\pi}$ is proved via safety of the **policy predicate abstraction** of $\Theta^{\pi}$, $\Theta^{\pi}_{\mathcal{P}} = \langle \mathcal{S}_{\mathcal{P}}, \mathcal{L}, \mathcal{T}^{\pi}_{\mathcal{P}} \rangle$ where $\mathcal{T}^{\pi}_{\mathcal{P}} = \{(s|_{\mathcal{P}}, l, s'|_{\mathcal{P}}) \mid (s, l, s') \in \mathcal{T}, \pi(s) = l\}$.

Vea (2022) introduce an algorithm to compute the fragment of $\Theta^{\pi}_{\mathcal{P}}$ reachable from the start condition $\phi_0$. The abstract transition problems in this computation involve checking whether $\pi$ selects the correct label: $(s_{\mathcal{P}}, l, s'_{\mathcal{P}}) \in \mathcal{T}_{\mathcal{P}}$ iff there exists an $l$-labeled operator $o \in \mathcal{O}$ and transition witness $s_w \in [s_{\mathcal{P}}]$, such that $s_w \models o$, $s_w[\![o]\!] \in [s'_{\mathcal{P}}]$ **and** $\pi(s_w) = l$. This is a key new source of complexity as the SMT sub-formula representing the neural network $\pi$ contains one disjunction for every neuron. Vea show how this can be dealt with through approximate SMT checks. In particular, this pertains to continuous relaxations of the integer state variables, solved with *Marabou* (Katz et al. 2019), an SMT solver tailored to NN analysis.

### CEGAR

To find a suitable predicate set $\mathcal{P}$ automatically, Vea (2023) employ counterexample-guided abstraction refinement (CEGAR) (Clarke et al. 2003). Starting from an initially coarse predicate set $\mathcal{P} = \emptyset$, they iteratively search for an abstract unsafe path $\sigma_{\mathcal{P}}$ in $\Theta^{\pi}_{\mathcal{P}}$. If $\sigma_{\mathcal{P}}$ is spurious, i.e., without concretization in $\Theta^{\pi}$, they refine $\mathcal{P}$ by adding predicates based on the source of spuriousness in $\sigma_{\mathcal{P}}$, and iterate. If $\sigma_{\mathcal{P}}$ is not spurious then $\pi$ is proved unsafe. Conversely, if no $\sigma_{\mathcal{P}}$ exists then $\pi$ is proved safe.

For refinement, Vea distinguish two sources of spuriousness, *over-approximation of the transition behavior* and spuriousness *induced by the policy*, Figure 1 shows an illustration. Consider a non-spurious prefix $\sigma^{i}_{\mathcal{P}} = \langle s^{0}_{\mathcal{P}}, l^{0}, \ldots, l^{i-1}, s^{i}_{\mathcal{P}} \rangle$ of $\sigma_{\mathcal{P}}$, i.e., there exists a prefix concretization $\langle s^{0}, l^{0}, \ldots, l^{i-1}, s^{i} \rangle$ in $\Theta^{\pi}$. Moreover, say $\sigma_{\mathcal{P}}$ next proceeds to $s^{i+1}_{\mathcal{P}}$ via label $l^{i}$. This is possible due to some transition witness $s^{i}_{w} \in [s^{i}_{\mathcal{P}}]$ with outgoing transition $(s^{i}_{w}, l^{i}, s^{i+1})$ such that $s^{i+1} \in [s^{i+1}_{\mathcal{P}}]$. Spuriousness due to **over-approximation of the transition behavior** occurs, if no such witness is reachable under concretizations of $\sigma^{i}_{\mathcal{P}}$, i.e., there does not exist a concrete path $\langle s^{0}, l^{0}, \ldots, l^{i-1}, s^{i}_{w} \rangle$. This source of spuriousness also occurs in standard PA. Here, Vea refine based on weakest precondition computation over $\langle s^{0}_{\mathcal{P}}, l^{0}, \ldots, l^{i-1}, s^{i}_{\mathcal{P}} \rangle$ (cf. (Podelski and Rybalchenko 2007)).

In PPA, we additionally need that $\pi(s^{i}_{w}) = l^{i}$, i.e., $\pi$ selects the respective action label in the witness state. If there exists a state $s^{i} \in [s^{i}_{\mathcal{P}}]$ with $(s^{i}, l^{i}, s^{i+1})$ (i.e., a witness in standard PA) reachable via concretizations, but $\pi(s^{i}) \neq l^{i}$ for any such $s^{i}$, then the spuriousness is **induced by the policy**. For refinement of policy-induced spuriousness, Vea apply a technique they baptize **witness splitting**. They consider an individual concretization $\langle s^{0}, l^{0}, \ldots, l^{i-1}, s^{i} \rangle$ of $\sigma^{i}_{\mathcal{P}}$ for which action label selection deviates, $\pi(s^{i}) \neq l^{i}$, and introduce predicates that enable to distinguish $s^{i}$ and $s^{i}_{w}$ (the witness state found by PPA). Intuitively, these predicates approximate the decision boundary of $\pi$ within $s^{i}_{\mathcal{P}}$ by the difference between $s^{i}_{w}$ and $s^{i}$, *spitting* $s^{i}_{\mathcal{P}}$ in the refined abstraction. Vea show that, while a spurious counterexample may persist after a single refinement step based on this approximation, it is guaranteed to be removed within finitely many refinement steps, and hence the approach is complete.

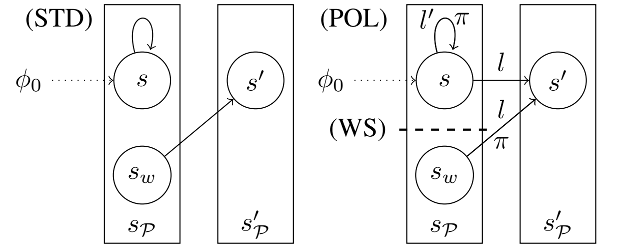

Figure 1: Illustrations (adapted from Vea (2023)) for standard spuriousness (STD), policy-induced spuriousness (POL) and refinement based on witness splitting (WS). $s \in [s_{\mathcal{P}}]$ is a concrete state reachable from $\phi_0$ under (prefix) concretizations of a given abstract path $\sigma_{\mathcal{P}}$. $s_w \in [s_{\mathcal{P}}] \setminus \{s\}$ is a witness of the next abstract transition $(s_{\mathcal{P}}, l, s'_{\mathcal{P}})$ of $\sigma_{\mathcal{P}}$. The dashed line indicates the split between $s$ and $s_w$ obtained by adding new predicates.

### Open Questions

Vea demonstrate, on a collection of benchmarks adapted from AI planning, that this approach tends to outperform encodings into the state-of-the-art non-NN-tailored verification tool NUXMV (Cavada et al. 2014), in particular being the only approach that succeeds in proving policies safe. These results are encouraging. However, in their policy refinement, they disregard the question of which aspects (i.e., state variables) are actually relevant to $\pi$'s decision in $s^{i}_{w}$. Instead, they introduce one box constraint for each variable $v \in \mathcal{V}$: $v \leq s^{i}_{w}(v)$ if $s^{i}_{w}(v) < s^{i}(v)$ and $v \geq s^{i}_{w}(v)$ if $s^{i}_{w}(v) > s^{i}(v)$.

This leaves us with the question whether performance can be improved via a more targeted selection of *(i) the variables* and *(ii) the split predicate* $v \leq c$. For (i), Vea's approach is complete as long as at least one variable is selected. For (ii), they split right *at the witness* $s^{i}_{w}(v)$, but, in principle, any *split point* $c$ between $s^{i}_{w}(v)$ and $s^{i}(v)$ can be chosen. In our work, we investigate these questions. Specifically, we (i) provide a variable selection technique inspired by ideas from formal XAI (Sec. 4) and (ii) experiment with different heuristics for selecting the split point (Sec. 5)).

For the remainder of this paper, we consider an abstract path prefix $\langle s^{0}_{\mathcal{P}}, l^{0}, \ldots, l^{i-1}, s^{i}_{\mathcal{P}} \rangle$ with policy-induced spuriousness in $s^{i}_{\mathcal{P}}$, prefix concretization $\langle s^{0}, l^{0}, \ldots, l^{i-1}, s^{i} \rangle$, and witness $s^{i}_{w} \in [s^{i}_{\mathcal{P}}]$. For readability, we drop the index $i$ and abbreviate $\pi(s^{i}_{w})$ by $l_w$.

# 4 Variable Selection for Policy Refinement

In this section, we provide a variable selection technique, that partitions the state variables $\mathcal{V}$ into a set of **relevant variables** $\mathcal{V}_{rel}$ and **irrelevant variables** $\mathcal{V}_{irr}$ for $\pi$ in $s_w$. The motivation is to perform witness splitting using the relevant subset $\mathcal{V}_{rel}$ only.

The technique is inspired by ideas from formal explainable AI, specifically VERIX (VERIfied eXplainability) (Wu, Wu, and Barrett 2022). VERIX computes a minimal subset of input features that *explains* the behavior of a neural network for a given input, such as an image in computer vision or a text in natural language processing. In essence, VERIX finds explanations by identifying sets of *irrelevant* features. Given a perturbation bound $\epsilon$, a set of features is irrelevant, if the neural model behavior is invariant against arbitrary $\epsilon$-bounded perturbations of all irrelevant features while fixing the other features, i.e., the explanation. If the set of irrelevant features is maximal, i.e., no feature can be added, the explanation is *optimal*. In other words, the features in the explanation are *relevant*.

In the following, we first spell out an adaption of VERIX to our witness splitting setting. Subsequently, we contribute a binary search framework BSEARCH, to get rid of the $\epsilon$ parameter. Their combination constitutes our variable selection technique.

## $\epsilon$-Relevant Variables

The notion of relevant and irrelevant features translates to state variables in an intuitive manner. In the witness splitting context, we are interested in perturbations "from" the witness $s_w$ "towards" the concretization state $s$ only. We maintain a relative perturbation bound $\epsilon \in [0, 1]$ and normalize the absolute perturbation bound for each variable to the interval between $s_w$ and $s$. That is, the value for variable $v$ in the *perturbed state* $\hat{s}$ is bounded by $|\hat{s}(v) - s_w(v))| \leq \epsilon \cdot |s(v) - s_w(v)|$ and $s_w(v) \leq \hat{s}(v)$ if $s_w(v) \leq s(v)$ respectively $s_w(v) \geq \hat{s}(v)$ if $s_w(v) \geq s(v)$; abbreviated $\hat{s}(v) \in D^{\epsilon}_{v}(s_w, s)$.

**Definition 1** ($\epsilon$-Relevant Variables)**.** Given states $s_w, s \in \mathcal{S}$ with $\pi(s_w) = l_w \neq \pi(s)$ and a perturbation bound $\epsilon \in [0, 1]$. $\mathcal{V}_{irr} \subseteq \mathcal{V}$ is **$\epsilon$-irrelevant** for $\pi$ in $s_w$ if for any state $\hat{s} \in \mathcal{S}$ such that $\hat{s}(v) \in D^{\epsilon}_{v}(s_w, s)$ for each $v \in \mathcal{V}_{irr}$ and $\hat{s}(v) = s_w(v)$ for each $v \in \mathcal{V} \setminus \mathcal{V}_{irr}$, it holds $\pi(\hat{s}) = l_w$.

$\mathcal{V}_{irr}$ is **maximal** if no superset of $\mathcal{V}_{irr}$ is irrelevant. If $\mathcal{V}_{irr}$ is maximal, its complement $\mathcal{V}_{rel} = \mathcal{V} \setminus \mathcal{V}_{irr}$ is **$\epsilon$-relevant**.

We provide a graphical illustration in Figure 2. Intuitively, a set of state variables $\mathcal{V}_{irr}$ is $\epsilon$-irrelevant if we can perturb $\mathcal{V}_{irr}$ in $s_w$ up to bound $\epsilon$ (while fixing $\mathcal{V} \setminus \mathcal{V}_{irr}$) with the policy still selecting $l_w$ in the perturbed state $\hat{s}$. If $\mathcal{V}_{irr}$ is maximal, i.e., no variable can be added, then its complement $\mathcal{V}_{rel} = \mathcal{V} \setminus \mathcal{V}_{irr}$ is $\epsilon$-relevant. We remark that Definition 1 is local in that for a given bound $\epsilon$ there may exist more than one $\epsilon$-relevant set $\mathcal{V}_{rel}$. An interesting problem would be to find $\mathcal{V}_{rel}$ with minimum size, which is, however, computationally extremely expensive. Following Wu et al. (2022), we instead settle for the computationally cheaper option of computing $\mathcal{V}_{rel}$ greedily given a fixed *traversal order* of the variables.

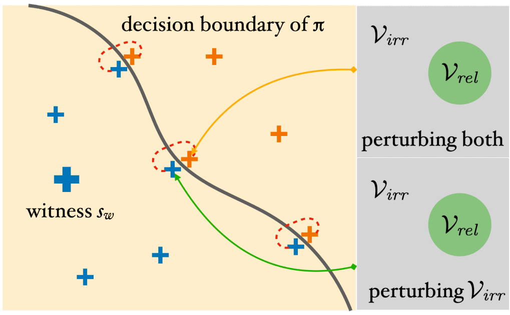

Figure 2: Graphical illustration of VERIX deployed in policy predicate abstraction, adapted from (Wu, Wu, and Barrett 2022). The big blue "+" denotes the witness state $s_w$, and the small "+"s are its variants with $\epsilon$-perturbations on state variables. Perturbing the irrelevant variables (blue "+") will definitely *not* change the action label, whereas perturbing both (orange "+") may lead to a label change. Particularly, when the irrelevant variable set $\mathcal{V}_{irr}$ is *maximal* (i.e., its complement $\mathcal{V}_{rel}$ is $\epsilon$-relevant) perturbing $\mathcal{V}_{irr}$ plus any single variable in $\mathcal{V}_{rel}$ can breach the decision boundary of the policy $\pi$, as shown by the variants connected by dotted lines.

**Algorithm 1:** VERIX (Wu, Wu, and Barrett 2022) (adapted to WS setting).

```
Input: States $s_w, s \in \mathcal{S}$ such that $\pi(s_w) = l_w \neq \pi(s)$.
Parameter: perturbation-bound $\epsilon \in [0, 1]$.
Output: $\epsilon$-relevant $\mathcal{V}_{rel} \subseteq \mathcal{V}$.
1  $(\mathcal{V}_{rel}, \mathcal{V}_{irr}) \leftarrow (\emptyset, \emptyset)$
2  for $v' \in$ TRAVERSALORDER$(s_w)$ do
3      $\mathcal{V}'_{irr} \leftarrow \mathcal{V}_{irr} \cup \{v'\}$
4      $\phi \leftarrow \bigwedge_{v \in \mathcal{V}'_{irr}} \hat{s}(v) \in D^{\epsilon}_{v}(s_w, s)$
5      $\phi \leftarrow \phi \bigwedge_{v \in \mathcal{V} \setminus \mathcal{V}'_{irr}} \hat{s}(v) = s_w(v)$
6      HOLD $\leftarrow$ CHECK$(\phi \Rightarrow \pi(\hat{s}) = l_w)$
7      if HOLD then
8          $\mathcal{V}_{irr} \leftarrow \mathcal{V}_{irr} \cup \{v'\}$
9      else
10         $\mathcal{V}_{rel} \leftarrow \mathcal{V}_{rel} \cup \{v'\}$
```

**VERIX Algorithm.** Algorithm 1 shows the adaption of the VERIX algorithm to our setting. We are given a witness $s_w$ and a concretization state $s$ with $\pi(s_w) = l_w \neq \pi(s)$ as well as a perturbation bound $\epsilon$. VERIX maintains two sets $\mathcal{V}_{irr}$ and $\mathcal{V}_{rel}$, initialized to $\emptyset$ (line 1). It then iterates each state variable $v'$ according to a traversal order (see below) fixed for $s_w$ (line 2), and checks whether $v'$ can be added to $\mathcal{V}_{irr}$. For this purpose, it constructs a formula $\phi$ over the perturbation state $\hat{s}$ where each $v \in \mathcal{V}_{irr}$ as well as $v'$ is $\epsilon$-bounded (line 4) and all remaining variables are fixed (line 5). It then checks via a call to a sub-procedure CHECK whether $\phi$ implies $\pi(\hat{s}) = l_w$ (line 6), i.e., whether $\pi$ selecting $l_w$ is invariant to perturbation to $\mathcal{V}_{irr} \cup \{v'\}$. In practice, CHECK can be instantiated with an off-the-shelf

SMT solver (e.g., *Marabou* (Katz et al. 2019)).[1] If CHECK holds, $\mathcal{V}_{irr} \cup \{v'\}$ is indeed irrelevant and $v'$ can be added to $\mathcal{V}_{irr}$ (line 8). If not, $\mathcal{V}_{rel}$ is added to $\mathcal{V}_{rel}$ (line 10). After all variables are traversed, each $v \in \mathcal{V}$ is placed in either of the two sets $\mathcal{V}_{irr}$ or $\mathcal{V}_{rel}$; $\mathcal{V}_{irr}$ is a maximal $\epsilon$-irrelevant set and $\mathcal{V}_{rel}$ is $\epsilon$-relevant.

**Theorem 2** (Correctness)**.** Algorithm 1 computes a $\epsilon$-relevant variable set $\mathcal{V}_{rel} \subseteq \mathcal{V}$.

Theorem 2 follows analogously to explanation *optimality* for original VERIX (Wu, Wu, and Barrett 2022). We provide a formal proof in the appendix.

**Traversal Order.** The traversal order can significantly affect the size and shape of $\mathcal{V}_{irr}$ respectively $\mathcal{V}_{rel}$. Following Wu et al. (2022), we use a $s_w$-dependent heuristic that aims to produce small $\mathcal{V}_{rel}$ based on **variable sensitivity**.

**Definition 3** (Variable Sensitivity)**.** The **variable sensitivity** for $v \in \mathcal{V}$ in $s_w \in \mathcal{S}$ is

$$sens(v) = \sum_{l \in \mathcal{L}} \left| \pi^l(s_w[v \mapsto up_v]) - \pi^l(s_w[v \mapsto lo_v]) \right|,$$

where $lo_v$ and $up_v$ are the lower and upper bound of $v$, $s_w[v \mapsto c]$ is the update of $s_w$ by $v \mapsto c$, and $\pi^l(s)$ is the value of the output layer neuron $l$ in the neural network $\pi$ given input $s$.

Variable sensitivity is inspired by *feature-level sensitivity* (Wu, Wu, and Barrett 2022). Intuitively, we measure how sensitive the policy $\pi$ is to perturbations of $v$ in $s_w$, comparing the output layer difference for maximal and minimal perturbation ($up_v$ and $lo_v$).

Given *sens*, TRAVERSALORDER($s_w$) iterates each state variable $v$ from least to most sensitive. This is expected to lead to larger $\mathcal{V}_{irr}$ and smaller $\mathcal{V}_{rel}$.

We remark that *sens* in isolation can already be used to select a subset of *heuristically-relevant* variables for witness splitting (e.g., the most sensitive variable only). We investigate this option as a baseline in our experiments.

## $\epsilon$-Minimal Relevant Variable Set

VERIX leaves the perturbation bound $\epsilon$ as a user-parameter to customize. In our case, however, we do not consider a specific variable selection problem $(s_w, s)$ in isolation. Instead, we want to repeatedly select a subset of relevant variables for changing pairs of $s_w$ and $s$ each time we refine policy-induced spuriousness throughout an invocation of CEGAR. Ultimately, we want $\epsilon$ to automatically adapt to the variable selection problem at hand.

We achieve this via a binary search around VERIX to find a minimal perturbation bound $\epsilon$ such that a non-empty $\epsilon$-relevant $\mathcal{V}_{rel}$ exists. Intuitively, this enables to select the subset of variables that is relevant to $\pi$'s decision *near* $s_w$.[2]

**Algorithm 2:** BSEARCH

```
  Input: States $s_w, s \in \mathcal{S}$ such that $\pi(s_w) = l_w \neq \pi(s)$.
  Parameter: $\epsilon$-precision $\tau \in (0, 1)$
  Output: $\epsilon$-minimal relevant $\mathcal{V}_{rel} \subseteq \mathcal{V}$.
1 $(\mathcal{V}_{rel}, \epsilon_{lo}, \epsilon, \epsilon_{up}) \leftarrow (\emptyset, 0, 0.5, 1)$
2 while $\epsilon_{up} - \epsilon_{lo} > \tau$ do
3     $\mathcal{V}'_{rel} \leftarrow$ VERIX$(s_w, s, \epsilon)$
4     if $\mathcal{V}'_{rel} = \emptyset$ then
5         $(\epsilon_{lo}, \epsilon, \epsilon_{up}) \leftarrow (\epsilon, \epsilon + (\epsilon_{up} - \epsilon)/2, \epsilon_{up})$
6     else
7         $(\mathcal{V}_{rel}, \epsilon_{lo}, \epsilon, \epsilon_{up}) \leftarrow (\mathcal{V}'_{rel}, \epsilon_{lo}, \epsilon_{lo} + (\epsilon - \epsilon_{lo})/2, \epsilon)$
8 if $\mathcal{V}_{rel} = \emptyset$ then $\mathcal{V}_{rel} \leftarrow$ VERIX$(s_w, s, 1)$
```

Algorithm 2 shows the pseudocode for the binary search BSEARCH. Like VERIX it is given a witness $s_w$ and a concretization state $s$ with $\pi(s_w) = l_w \neq \pi(s)$. However, unlike VERIX it is not given a specific perturbation $\epsilon$ but only a $\epsilon$-precision $\tau \in (0, 1)$ up to which a minimal $\epsilon$ is computed. The latter is required to guaranteed termination.

BSEARCH maintains a set $\mathcal{V}_{rel}$, a perturbation bound $\epsilon$, a lower bound $\epsilon_{lo}$ on $\epsilon$ and an upper bound $\epsilon_{up}$ on $\epsilon$. $\mathcal{V}_{rel}$ is initialized to $\emptyset$, $\epsilon_{lo}$ to the perturbation minimum 0, $\epsilon_{up}$ to the maximum 1, and $\epsilon$ to the middle point 0.5 (line 1). BSEARCH then iterates until the search interval $(\epsilon_{lo}, \epsilon_{up})$ becomes smaller than $\tau$ (line 2). In each iteration VERIX is called with the current perturbation bound $\epsilon$ (line 3).

If the $\epsilon$-relevant variable set $\mathcal{V}'_{rel}$ found by VERIX is empty, the search interval is updated to its upper half $(\epsilon, \epsilon_{up})$ and $\epsilon$ is increased to the new middle point accordingly (line 5). Intuitively, $\pi$'s decision is invariant under $\epsilon$-bounded perturbation to all variables. Hence, all variables are irrelevant and $\epsilon$ must be increased to find a non-empty relevant set.

If $\mathcal{V}'_{rel}$ is non-empty, it is assigned to $\mathcal{V}_{rel}$ as the new candidate of being a non-empty $\epsilon$-relevant set with minimal $\epsilon$ (line 7). The search interval is now updated to its lower half $(\epsilon_{lo}, \epsilon)$ and $\epsilon$ is decreased to the new middle point. That is, in the next iterations, BSEARCH continues to search for a smaller $\epsilon$ such that a non-empty $\epsilon$-relevant variable set exists. We remark that also after finding a candidate set $\mathcal{V}_{rel}$, the if-branch ($\mathcal{V}'_{rel} = \emptyset$) may be visited. Intuitively, BSEARCH approaches the smallest candidate bound found so far (i.e., $\epsilon_{up}$) from below.

Without $\epsilon$-precision $\tau > 0$ the loop in Algorithm 2 will never terminate. In either branch of the if-else-condition, there always exists a non-empty search interval ($(\epsilon, \epsilon_{up})$ or $(\epsilon_{lo}, \epsilon)$) to be explored. Hence, $\tau > 0$ is indeed required to guarantee termination. However, in practice, for reasonably small $\tau$, we expect the computed $\epsilon$ to be usually minimal without restriction.

Due to $\tau$-precision, the loop in Algorithm 2 may terminate, i.e., $(\epsilon_{lo}, \epsilon_{up})$ becomes smaller than $\tau$, while still $\mathcal{V}_{rel} = \emptyset$. This corner case is caught by computing the variable set that is relevant for $\epsilon = 1$ (line 8). Note that since $\pi(s_w) \neq \pi(s)$, the 1-relevant variable set is guaranteed to

[1] It is also possible to first run an adversarial attack algorithm to try to quickly falsify $\phi \Rightarrow \pi(\hat{s}) = l_w$ before deploying the SMT solver, but on the benchmarks that we study those SMT calls do not incur a large overhead.

[2] We remark that this optimization criteria is not equivalent to finding $\epsilon$ with non-empty $\mathcal{V}_{rel}$ that is minimal in size.

be non-empty. We remark that in our empirical study, this corner case never occurred.

**Theorem 4** (Correctness)**.** Algorithm 2 computes a non-empty $\epsilon$-relevant variable set $\mathcal{V}_{rel} \subseteq \mathcal{V}$ such that $\epsilon$ is minimal up to $\tau$.

To facilitate readability, we move the formal proof to the appendix.

*Proof sketch..* Let $\mathcal{V}_{rel}$ be the variable set computed by Algorithm 2. Case distinction.

- $\mathcal{V}_{rel}$ is set (non-empty) in the loop (line 7): Whenever $\mathcal{V}_{rel}$ is updated, $\epsilon_{up}$ is set to current $\epsilon$. Hence, $\mathcal{V}_{rel}$ is $\epsilon_{up}$-relevant at loop termination. $\epsilon_{lo}$ is 0 ($\emptyset$ is 0-relevant) or updated for $\mathcal{V}'_{rel} = \emptyset$ (line 5). Hence, there does not exist $\epsilon' \leq \epsilon_{lo}$ with a non-empty $\epsilon'$-relevant variable set. Since additionally $\epsilon_{up} - \epsilon_{lo} < \tau$, $\epsilon_{up}$ is minimal up to $\tau$.
- $\mathcal{V}_{rel}$ is set after loop termination (line 8): Since $\mathcal{V}_{rel} = \emptyset$ at loop termination,, we have $\epsilon_{up} = 1$. Analogously, to the first case, we have that there does not exist $\epsilon' \leq \epsilon_{lo}$ with a non-empty $\epsilon'$-relevant variable set. Since also $\epsilon_{up} - \epsilon_{lo} < \tau$, 1 is minimal up to $\tau$.

□

## 5 Split Predicate Selection

In the previous section, we have provided a technique to select a subset of relevant state variables $\mathcal{V}_{rel}$ for witness splitting. In this section, we propose different heuristics for selecting the split predicate $v \gtrless c$ for each $v \in \mathcal{V}_{rel}$. In principle, witness splitting can be applied using any split predicate $v \leq c$ with $s_w(v) \leq c < s(v)$ if $s_w(v) < s(v)$, and $v \geq c$ with $s_w(v) \geq c > s(v)$ if $s_w(v) > s(v)$, respectively.

- (AT-WITNESS) In their original approach, Vea split right *at the witness* $s_w(v)$ for each state variable $v$. That is, split predicate $v \leq s_w(v)$ if $s_w(v) < s(v)$ and $v \geq s_w(v)$ if $s_w(v) > s(v)$. Intuitively, this choice is motivated in that $s_w(v)$ is known to "lie" within the decision boundary of $\pi$ selecting the desired action label, namely $\pi(s_w)$. In contrast, due to the complex structure of NN, no adhoc predication on $\pi$'s behavior for split points $c$ beyond $s_w(v)$ can be made.
- (MID-SPLIT) A disadvantage of AT-WITNESS is that it may heavily underestimate the distance to the decision boundary from $s_w$ towards $s$. This may result in unnecessarily many iterations of $\pi$-induced refinement; stepwise approximating the decision boundary. To mitigate such behavior, we experiment with a heuristic that splits in the middle between $s_w$ and $s$. That is, $v \leq s_w(v) + \frac{1}{2} \cdot (s(v) - s_w(v))$ for $s_w(v) < s(v)$ (and analogously for $s_w(v) > s(v)$). While there is no guarantee in terms of $c$'s relation to $\pi$'s decision boundary, middle-point based approximations can be observed to perform well in various contexts.
- ($\epsilon$-SPLIT) Our variable selection technique computes, in addition to the relevant variable subset $\mathcal{V}_{rel}$, also a corresponding perturbation bound $\epsilon$. Broadly speaking, $\epsilon$ can be considered as an estimate of the decision boundary of $\pi$: $\epsilon$-bounded perturbation to any $v \in \mathcal{V}_{rel}$ (plus $\mathcal{V}_{irr}$) allows for perturbed states $\hat{s}$ such that $\pi(\hat{s}) \neq \pi(s_w)$; crossing the decision boundary. We, hence, experiment with a heuristic that uses a split point relative to $\epsilon$. That is, $v \leq s_w(v) + \epsilon \cdot (s(v) - s_w(v))$ for $s_w(v) < s(v)$ (and analogously for $s_w(v) > s(v)$).

## 6 Experiments

We implemented our approach on top of Vea's C++ code base, which uses $Z3$ (de Moura and Bjørner 2008) and *Marabou* (Katz et al. 2019) to solve the various SMT queries during PPA computation and refinement. For CHECK (Algorithm 1, line 6) we query *Marabou*.[3] All experiments were run on machines with Intel Xenon E5- 2650 processors at 2.2 GHz, with time and memory limits of 12 h and 4 GB. We next describe the experiments setup, then summarize our results.

### Experimental Setup

**Configurations.** We compare a broad range of algorithmic configurations for refinement of policy-induced spuriousness during CEGAR combining different variable selection techniques and split point heuristics.

- *Vea*$^{w}$ is the default configuration used by Vea, i.e., selecting all state variables[4] and using AT-WITNESS for splitting.
- *Vea*$^{mid}$ still selects all variables but uses MID-SPLIT for splitting.
- *BsVerix*$^{w}$ applies the variable selection technique introduced in Section 4, i.e., BSEARCH around VERIX. For splitting it uses Vea's default AT-WITNESS. For *BsVerix*$^{w}$, and all following configurations using BSEARCH, we set $\epsilon$-precision $\tau = 0.05$.
- *BsVerix*$^{mid}$ also applies variable selection, though, using MID-SPLIT for splitting.
- *BsVerix*$^{\epsilon}$ applies variable selection and uses $\epsilon$-SPLIT for splitting.
- *sens*$^{mid}$ uses $sens$ (cf., Definition 3) to select the most-sensitive variable only, splitting based on MID-SPLIT. It serves as a simple baseline to the more advanced variable selection based on VERIX.
- As the ultimate baseline, *Rand* selects a random number of variables and a random split point between $s_w(v)$ and $s(v)$ for each selected $v$.

**Randomization.** The underlying CEGAR refinement approach involves two sources of non-determinism.

- Each refinement step is dependent on the detected spurious counterexample. Depending on the order in which the abstract state space is explored, different counterexamples, i.e., abstract unsafe paths, may be found. Vea provide a simple hamming distance heuristic, greedily

[3] Our tool (and all experiments) are available at request.

[4] State variables $v \in \mathcal{V}$ such that $s_w(v) = s(v)$ are always ignored, since no split point is possible.

| Benchmark | NN | Safe | Time | | | | | | | ($Vea^w$, $BsVerix^w$) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | $Vea^w$ | $Vea^{mid}$ | $BsVerix^w$ | $BsVerix^{mid}$ | $BsVerix^\epsilon$ | $sens^{mid}$ | *Rand* | #It | $\pi$-It | $\|\mathcal{P}\|$ | $\pi$-$\|\mathcal{P}\|$ |
| | 16 | ✓ | 5.1 · 0±−1 | **4.1 ·** 0±−1 | 6.6 · 0±−1 | 8.3 · 0±0 | 6.8 · 0±0 | 8.6 · 0±0 | 6.0 · 0±0 | (11, 14) | (3, 4) | (21, 20) | (10, 5) |
| 4 Blocks | 32 | ✓ | **1.9 ·** 1±0 | 2.5 · 1±0 | 3.0 · 1±1 | 3.5 · 1±1 | 3.4 · 1±1 | 3.6 · 1±1 | 3.3 · 1±0 | (11, 16) | (3, 4) | (21, 21) | (10, 4) |
| (cost-ign) | 64 | ✓ | **1.6 ·** 2±1 | 1.6 · 2±1 | 2.8 · 2±2 | 2.6 · 2±1 | 2.8 · 2±1 | 2.6 · 2±1 | 1.8 · 2±1 | (11, 14) | (2, 3) | (20, 19) | (6, 3) |
| | 16 | ✓ | 9.2 · 1±1 | 2.8 · 3±3 | **7.5 ·** 1±1 | 9.8 · 1±1 | 9.2 · 1±1 | 2.4 · 2±2 | 1.8 · 2±2 | (21, 30) | (4, 8) | (38, 38) | (18, 10) |
| 6 Blocks | 32 | ✓ | 1.5 · 2±1 | 1.5 · 2±1 | 1.5 · 2±1 | 1.5 · 2±1 | 1.5 · 2±1 | **1.4 ·** 2±1 | 1.6 · 2±1 | (20, 32) | (6, 11) | (38, 39) | (20, 11) |
| (cost-ign) | 64 | ✓ | **1.3 ·** 4±3 | 1.7 · 4±3 | 1.8 · 4±3 | 1.9 · 4±4 | 2.1 · 4±3 | 3.2 · 4±4 | 1.7 · 4±3 | (21, 31) | (4, 8) | (35, 38) | (12, 8) |
| | 16 | ✓ | 1.3 · 4±4 | 1.4 · 4±4 | **4.2 ·** 3±3 | 2.3 · 4±4 | 1.3 · 4±4 | 1.3 · 4±4 | 4.0 · 4±4 | (36, 52) | (6, 10) | (62, 62) | (27, 11) |
| 8 Blocks | 32 | ✓ | 8.0 · 4±4 | - | 8.0 · 4±4 | - | **7.9 ·** 4±4 | 8.0 · 4±4 | - | (37, 52) | (6, 10) | (63, 65) | (23, 13) |
| (cost-ign) | 64 | ? | - | - | - | - | - | - | - | - | - | - | - |
| | 16 | × | 4.2 · 1±1 | 3.9 · 1±1 | 9.0 · 1±1 | **3.8 ·** 1±1 | 3.9 · 1±1 | 5.5 · 1±1 | 4.4 · 1±1 | (24, 40) | (13, 28) | (48, 48) | (34, 31) |
| 8-puzzle | 32 | ✓ | 3.2 · 4±4 | 3.3 · 4±4 | 1.9 · 4±4 | 3.3 · 4±4 | 4.3 · 4±4 | 4.2 · 4±4 | **1.7 ·** 4±4 | (72, 88) | (7, 20) | (126, 128) | (29, 28) |
| (cost-ign) | 64 | ✓ | 6.5 · 4±4 | **4.9 ·** 4±4 | 6.5 · 4±4 | 7.9 · 4±4 | 6.1 · 4±4 | 5.0 · 4±4 | 6.3 · 4±4 | (58, 83) | (10, 31) | (116, 114) | (38, 35) |
| | 16 | ✓ | 1.5 · 1±1 | 2.0 · 1±0 | 1.7 · 1±0 | 2.2 · 1±1 | 1.5 · 1±0 | **1.2 ·** 1±0 | 3.8 · 1±1 | (12, 18) | (5, 7) | (25, 24) | (14, 8) |
| 4 Blocks | 32 | ✓ | 1.4 · 3±2 | 2.0 · 3±3 | 1.0 · 4±4 | **6.3 ·** 2±2 | 8.9 · 2±2 | 8.1 · 2±3 | 3.7 · 3±3 | (13, 19) | (4, 10) | (27, 26) | (14, 11) |
| (cost-awa) | 64 | ✓ | 6.9 · 4±4 | 7.6 · 4±4 | **6.5 ·** 4±4 | 8.2 · 4±4 | 8.2 · 4±4 | 8.2 · 4±4 | - | (9, 16) | (3, 8) | (24, 21) | (15, 8) |
| | 16 | ✓ | 9.5 · 2±3 | 1.7 · 3±3 | 1.7 · 3±3 | 2.3 · 3±3 | 1.9 · 3±3 | **7.1 ·** 2±3 | 3.7 · 3±3 | (20, 31) | (6, 10) | (48, 45) | (30, 17) |
| 6 Blocks | 32 | ✓ | 3.2 · 4±4 | 2.3 · 4±4 | 8.2 · 3±3 | 6.3 · 3±3 | **4.1 ·** 3±3 | 6.0 · 3±3 | 1.6 · 4±4 | (21, 31) | (6, 7) | (46, 41) | (26, 10) |
| (cost-awa) | 64 | ? | - | - | - | - | - | - | - | - | - | - | - |

Table 1: Runtime results in seconds for the evaluated CEGAR configurations ($Vea^w$, $Vea^{mid}$, $BsVerix^w$, $BsVerix^{mid}$, $BsVerix^\epsilon$, $sens^{mid}$, *Rand*) for different benchmarks and NN policies averaged over 10 randomized invocations, where each result $t \cdot \mathtt{i} + \mathtt{j}$ specifies the mean $t \cdot 10^i$ and the magnitude of the observed standard deviation $10^j$. Timeouts (exceeding the time limit of 12 h) are weighted as 24h. - indicates timeouts for all invocations. For $Vea^w$ and $BsVerix^w$ we additionally compare the number of CEGAR iterations (#It), the number of policy-induced refinement steps ($\pi$-It), the size of the final predicate set ($|\mathcal{P}|$), the number of predicates added during policy-induced refinement ($\pi$-$|\mathcal{P}|$) – averaged over 10 randomized invocations.

exploring abstract states for which the distance to an unsafe state is estimated to be small, to speed up the detection of unsafe paths. However, distinct abstract states may obtain the same heuristic estimate. In this case, Vea break ties non-deterministically. To account for this source of non-determinism and to investigate the impact of different spurious counterexamples, we invoke each of the configurations above with multiple seeds and break ties randomly.

- For policy-induced spuriousness in particular, each refinement step is also influenced by the deployed witness $s_w$ – an abstract transition may have multiple witnesses – as well as its counterpart $s$ in the deployed prefix concretization – there may be multiple prefix concretizations. Both of these states are extracted from the underlying SMT solvers (*Marabou* and $Z3$) as solutions found for the problem at hand (PPA transition problem and prefix concretization). To account for this additional source of non-determinism – the SMT solver is essentially a black-box – we also invoke each configuration with different seeds for $Z3$ and *Marabou*.

In total, we invoke each configuration with 10 different seeds.

**Benchmarks.** Due to the large configuration space (variable & split point selection compounded with randomization), we focus on a subset of Vea's benchmarks. To give a brief overview, the benchmarks are variants of the planning domains Blocksworld and SlidingTiles. Actions moving a block/tile $x$ may probabilistically fail, in which case the cost of moving $x$ (represented by a state variable) is incremented. These probabilistic transitions (over which the policy is learned) are abstracted into non-deterministic ones for the purpose of verification, amounting to a worst-case analysis. In both domains, the start conditions impose a partial order on the block/tile positions. In Blocksworld, a state is unsafe if the number of blocks on the table exceeds a fixed limit. In SlidingTiles, unsafe states are specified in terms of a set of unsafe tile positions.

For each domain instance, there are three NN policies trained by Vea using Q-learning (Mnih et al. 2015), each with two hidden layers of size 16, 32, respectively 64. There are policies that do, and ones that do not, take move costs into account.

## Experimental Results

Table 1 shows our results. On 16 out of the 18 problem instances at least one configuration wins. The conclusion are:

**Variable Selection.** Policy-specific variable selection (*BsVerix*, $sens^{mid}$) can improve performance over the default applied by Vea. However, there is no clear winning configuration. The *BsVerix*-based configurations ($BsVerix^w$, $BsVerix^{mid}$, $BsVerix^\epsilon$) have at least one best performing configuration (*smallest mean*) on 7 problem instances, while the configurations selecting all variables ($Vea^w$, $Vea^{mid}$) win on 5 instances. On 3 instances, the baseline based on variable sensitivity ($sens^{mid}$) performs best. The baseline based on random selection (*Rand*) still achieves best performance on one instance. On 9 instances, all configurations – including *Rand* – achieve performance in the same magnitude as the best performing configuration. Moreover, for each configuration we observe a standard deviation in the same order of magnitude as the average runtime on at least 8 out of the 16 problem instances.

In summary, these results are highly inconclusive. While the high standard deviation hints at a certain instability of the approach – presumably due to the policy-induced refinement using witness states extracted from the underlying SMT solvers – the performance of *Rand* may, at least to some degree, indicate that the current benchmark basis does not address the problem at sufficient scale.

**Split Predicate Selection.** Focusing on split predicate selection, the picture becomes less opaque. AT-WITNESS is the overall better performing heuristic (compared to MID-SPLIT and $\epsilon$-SPLIT). $Vea^{w}$ outperforms $Vea^{mid}$ on 11 problem instances – covering one instance on which $Vea^{mid}$ does not terminate. $BsVerix^{w}$ outperforms $BsVerix^{mid}$ and $BsVerix^{\epsilon}$ on 8 instances – $BsVerix^{mid}$ and $BsVerix^{\epsilon}$ only dominating on 4 respectively 3 instances. These results indicate that overall the more conservative split point estimate of AT-WITNESS turns out to be more stable. That said, there still is no clear winner. *Vea* configurations and *BsVerix* configuration both perform in the same order of magnitude on more than 10 instances.

***Vea* vs. *BsVerix*** While there is no clear picture runtime-wise, the impact of policy-specific variable selection can be clearly measured in terms of policy-induced refinement steps and the number of predicates added during these steps. Table 1 illustrates this by comparing $Vea^{w}$ and $BsVerix^{w}$. As intended, for $BsVerix^{w}$ the predicate selection becomes more cautious – fewer predicates $\pi$-$|\mathcal{P}|$ are added during policy-induced refinement steps while the number of refinement steps $\pi$-It grows. The latter also results in an increase of the total number of CEGAR iterations (#It). Interestingly, the total number of predicates $|\mathcal{P}|$ is usually similar in size. That is, the decrease in the number of predicates added during $\pi$-refinement $\pi$-$|\mathcal{P}|$ results in an increase of the number of predicates added for refinement of standard spuriousness. In accordance with this intuition, the increase in #It tends to be larger than the increase in $\pi$-It. For other *Vea*-*BsVerix*-pairs similar observations can be made.

## 7 Conclusion

Neural policy verification via policy predicate abstraction involves intricate problems in the choice of abstraction predicates, which naturally, for spuriousness caused by policy decisions, should target the neural decision boundary. We have introduced and evaluated a range of methods operationalizing this intuition. For the time being, while improvements over the previous simpler predicate selection method can be obtained, the results are largely inconclusive in that there is no winning configuration. To a degree at least, this is probably due to the current benchmark basis, which does not seem to exhibit the problem addressed at sufficient scale – a strong indication being that, in more than half of the benchmark instances, a random strategy achieves average performance in the same order of magnitude as the respective winning configuration. Future work will have to expand the benchmark basis, and reveal whether there exist more consistently well-performing configurations (potentially also based on different ideas, like minimum-distance states on the other side of the decision boundary as illustrated in Figure 2 (Croce and Hein 2020; Madry et al. 2018; Szegedy et al. 2014)). Portfolio methods are of interest too, as they might manage to reap the benefits of different strategies across benchmarks.

## Acknowledgments

This work was funded by DFG Grant 389792660 as part of TRR 248 – CPEC (https://perspicuous-computing.science). This work has received funding from the European Union's Horizon Europe Research and Innovation program under the grant agreement TUPLES No 101070149.

## References

Akintunde, M.; Lomuscio, A.; Maganti, L.; and Pirovano, E. 2018. Reachability Analysis for Neural Agent-Environment Systems. In Thielscher, M.; Toni, F.; and Wolter, F., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018, Tempe, Arizona 30 October - 2 November 2018*, 184–193. AAAI Press.

Akintunde, M. E.; Kevorchian, A.; Lomuscio, A.; and Pirovano, E. 2019. Verification of RNN-Based Neural Agent-Environment Systems. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii USA, January 27 - February 1, 2019*, 6006–6013. AAAI Press.

Amir, G.; Schapira, M.; and Katz, G. 2021. Towards Scalable Verification of Deep Reinforcement Learning. In *Formal Methods in Computer Aided Design, FMCAD 2021, New Haven, CT, USA, October 19-22, 2021*, 193–203. IEEE.

Ball, T.; Majumdar, R.; Millstein, T. D.; and Rajamani, S. K. 2001. Automatic Predicate Abstraction of C Programs. In Burke, M.; and Soffa, M. L., eds., *Proceedings of the 2001 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Snowbird, Utah, USA, June 20-22, 2001*, 203–213. ACM.

Barrett, C. W.; and Tinelli, C. 2018. Satisfiability Modulo Theories. In Clarke, E. M.; Henzinger, T. A.; Veith, H.; and Bloem, R., eds., *Handbook of Model Checking*, 305–343. Springer.

Cavada, R.; Cimatti, A.; Dorigatti, M.; Griggio, A.; Mariotti, A.; Micheli, A.; Mover, S.; Roveri, M.; and Tonetta, S. 2014. The nuXmv Symbolic Model Checker. In Biere, A.; and Bloem, R., eds., *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 *LNCS*, 334–342. Springer.

Clarke, E.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2003. Counterexample-guided abstraction refinement for symbolic model checking. *JACM*, 50(5): 752–794.

Croce, F.; and Hein, M. 2020. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proc. Mach. Learn. Res.*, 2206–2216. PMLR.

de Moura, L.; and Bjørner, N. 2008. Z3: An Efficient SMT Solver. In Ramakrishnan, C. R.; and Rehof, J., eds., *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008 Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *LNCS*, 337–340. Springer.

Dutta, S.; Chen, X.; and Sankaranarayanan, S. 2019. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In Ozay, N.; and Prabhakar, P., eds., *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, Montreal, QC, Canada, April 16-18, 2019*, 157–168. ACM.

Garg, S.; Bajpai, A.; and Mausam. 2019. Size Independent Neural Transfer for RDDL Planning. In Benton, J.; Lipovetzky, N.; Onaindia, E.; Smith, D. E.; and Srivastava, S., eds., *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15 2019*, 631–636. AAAI Press.

Graf, S.; and Saïdi, H. 1997. Construction of Abstract State Graphs with PVS. In Grumberg, O., ed., *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings*, volume 1254 of *LNCS*, 72–83. Springer.

Groshev, E.; Goldstein, M.; Tamar, A.; Srivastava, S.; and Abbeel, P. 2018. Learning Generalized Reactive Policies Using Deep Neural Networks. In de Weerdt, M.; Koenig, S.; Röger, G.; and Spaan, M. T. J., eds., *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018*, 408–416. AAAI Press.

Gupta, A.; and Strichman, O. 2005. Abstraction Refinement for Bounded Model Checking. In Etessami, K.; and Rajamani, S. K., eds., *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*, volume 3576 of *LNCS*, 112–124. Springer.

Henzinger, T. A.; Jhala, R.; Majumdar, R.; and McMillan, K. L. 2004. Abstractions from proofs. In Jones, N. D.; and Leroy, X., eds., *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14-16, 2004*, 232–244. ACM.

Huang, S.; Fan, J.; Li, W.; Chen, X.; and Zhu, Q. 2019. ReachNN: Reachability analysis of neural-network controlled systems. *ACM Trans. Embed. Comput. Syst.*, 18(5s): 106:1–106:22.

Issakkimuthu, M.; Fern, A.; and Tadepalli, P. 2018. Training Deep Reactive Policies for Probabilistic Planning Problems. In de Weerdt, M.; Koenig, S.; Röger, G.; and Spaan, M. T. J., eds., *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018*, 422–430. AAAI Press.

Ivanov, R.; Carpenter, T. J.; Weimer, J.; Alur, R.; Pappas, G. J.; and Lee, I. 2021. Verifying the Safety of Autonomous Systems with Neural Network Controllers. *ACM Trans. Embed. Comput. Syst.*, 20(1): 7:1–7:26.

Katz, G.; Barrett, C. W.; Dill, D. L.; Julian, K.; and Kochenderfer, M. J. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In Majumdar, R.; and Kuncak, V., eds., *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, volume 10426 of *LNCS*, 97–117. Springer.

Katz, G.; Huang, D. A.; Ibeling, D.; Julian, K.; Lazarus, C.; Lim, R.; Shah, P.; Thakoor, S.; Wu, H.; Zeljic, A.; Dill, D. L.; Kochenderfer, M.; and Barrett, C. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In Dillig, I.; and Tasiran, S., eds., *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *LNCS*, 443–452. Springer.

Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; and Vladu, A. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M. A.; Fidjeland, A.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nat.*, 518(7540): 529–533.

Podelski, A.; and Rybalchenko, A. 2007. ARMC: The Logical Choice for Software Model Checking with Abstraction Refinement. In Hanus, M., ed., *Practical Aspects of Declarative Languages, 9th International Symposium, PADL 2007, Nice, France, January 14-15, 2007*, volume 4354 of *LNCS*, 245–259. Springer.

Stahlberg, S.; Bonet, B.; and Geffner, H. 2022. Learning General Optimal Policies with Graph Neural Networks: Expressive Power, Transparency, and Limits. In Kumar, A.; Thiébaux, S.; Varakantham, P.; and Yeoh, W., eds., *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022, Singapore (virtual), June 13-24 2022*, 629–637. AAAI Press.

Sun, X.; Khedr, H.; and Shoukry, Y. 2019. Formal Verification of Neural Network Controlled Autonomous Systems. In Ozay, N.; and Prabhakar, P., eds., *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, Montreal, QC, Canada, April 16-18, 2019*, 147–156. ACM.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I. J.; and Fergus, R. 2014. Intriguing properties of neural networks. In Bengio, Y.; and LeCun, Y., eds., *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2503–2511.

Toyer, S.; Thiébaux, S.; Trevizan, F. W.; and Xie, L. 2020. ASNets: Deep Learning for Generalised Planning. *JAIR*, 68: 1–68.

Tran, H.; Cai, F.; Lopez, D. M.; Musau, P.; Johnson, T. T.; and Koutsoukos, X. D. 2019. Safety Verification of Cyber-Physical Systems with Reinforcement Learning Control. *ACM Trans. Embed. Comput. Syst.*, 18(5s): 105:1–105:22.

Vinzent, M.; Sharma, S.; and Hoffmann, J. 2023. Neural Policy Safety Verification via Predicate Abstraction: CEGAR. In *Thirty-Seventh AAAI Conference on Artificial Intelligence*. AAAI Press.

Vinzent, M.; Steinmetz, M.; and Hoffmann, J. 2022. Neural Network Action Policy Verification via Predicate Abstraction. In Kumar, A.; Thiébaux, S.; Varakantham, P.; and Yeoh, W., eds., *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling, ICAPS 2022, Singapore (virtual), June 13-24 2022*. AAAI Press.

Wu, M.; Wu, H.; and Barrett, C. 2022. VeriX: Towards Verified Explainability of Deep Neural Networks. *arXiv preprint arXiv:2212.01051.*

# A Proofs

In this section, we attach the proofs of the correctness results stated in the main text.

## Proofs for VeriX

We first prove **Theorem 2**, i.e., correctness of Algorithm 1 (VeriX). The presented proof is a translation of the proof for explanation optimality of original VeriX (Wu, Wu, and Barrett 2022) (Theorem 3.4) adapted to the witness splitting context.

*Proof.* By contradiction. Assumption ($\mathbb{A}$): $\mathcal{V}_{rel}$ as computed by Algorithm 1 is not $\epsilon$-relevant. That is, there exists a state variable $v' \in \mathcal{V}_{rel}$ such that for any perturbed state $\hat{s} \in \mathcal{S}$ with $\hat{s}(v) = s_w(v)$ for each $v \in \mathcal{V}_{rel} \setminus \{v'\}$ and $\hat{s}(v) \in D_v^\epsilon(s_w, s)$ for each $v \in \mathcal{V}_{irr} \cup \{v'\}$ (where $\mathcal{V}_{irr} = \mathcal{V} \setminus \mathcal{V}_{rel}$) it holds $\pi(\hat{s}) = l_w$.

Suppose Algorithm 1 examines $v'$ in the $i$th iteration. Let $\mathcal{V}_{irr}^i$ denote the irrelevant variable set in iteration $i$. By construction, we have $\mathcal{V}_{irr}^i \subseteq \mathcal{V}_{irr}$ and therewith $\mathcal{V}_{irr}' = \mathcal{V}_{irr}^i \cup \{v\} \subseteq \mathcal{V}_{irr} \cup \{v\}$. Therefore, any $\hat{s} \in \mathcal{S}$ that fulfills $\phi$ as per (line 4 - 5) also fulfills the premise of assumption $\mathbb{A}$. Hence, Check($\phi \Rightarrow \pi(\hat{s}) = l_w$) holds (line 6) and $v$ is added to the irrelevant variable set (line 8). This contradicts $v \in \mathcal{V}_{rel}$. □

## Proofs for BSearch

We now prove the correctness of Algorithm 2 (BSearch).

**Termination.** For sake of completeness, we first establish a termination guarantee for the binary search loop.

**Lemma A.** After $i$ iterations of Algorithm 2 it holds $\epsilon_{up} - \epsilon_{lo} = (\frac{1}{2})^i$ and $\epsilon = \epsilon_{lo} + \frac{\epsilon_{up} - \epsilon_{lo}}{2}$.

*Proof.* By induction. Induction base ($n = 0$): trivial.

Induction step ($i \rightarrow i + 1$): Let $\epsilon^i, \epsilon_{lo}^i, \epsilon_{up}^i$ denote the respective value after $i$ iterations. By induction hypothesis $\epsilon_{up}^i - \epsilon_{lo}^i = (\frac{1}{2})^i$ and $\epsilon^i = \epsilon_{lo}^i + \frac{\epsilon_{up}^i - \epsilon_{lo}^i}{2}$. Case distinction.

If Algorithm 2 sets $\epsilon_{lo}^{i+1} = \epsilon^i$ and $\epsilon_{up}^{i+1} = \epsilon_{up}^i$ (line 5), $\epsilon_{up}^{i+1} - \epsilon_{lo}^{i+1} = \epsilon_{up}^i - \epsilon^i = \epsilon_{up}^i - (\epsilon_{lo}^i + \frac{\epsilon_{up}^i - \epsilon_{lo}^i}{2}) = \frac{\epsilon_{up}^i - \epsilon_{lo}^i}{2} = \frac{1}{2} \cdot (\frac{1}{2})^i = (\frac{1}{2})^{i+1}$. Moreover, $\epsilon^{i+1} = \epsilon^i + \frac{\epsilon_{up}^i - \epsilon^i}{2} = \epsilon_{lo}^{i+1} + \frac{\epsilon_{up}^{i+1} - \epsilon_{lo}^{i+1}}{2}$ and thus the statement holds.

Else $\epsilon_{lo}^{i+1} = \epsilon_{lo}^i$ and $\epsilon_{up}^{i+1} = \epsilon^i$ (line 7) and hence $\epsilon_{up}^{i+1} - \epsilon_{lo}^{i+1} = (\epsilon_{lo}^i + \frac{\epsilon_{up}^i - \epsilon_{lo}^i}{2}) - \epsilon_{lo}^i = \frac{\epsilon_{up}^i - \epsilon_{lo}^i}{2} = (\frac{1}{2})^{i+1}$. Moreover, $\epsilon^{i+1} = \epsilon_{lo}^i + \frac{\epsilon^i - \epsilon_{lo}^i}{2} = \epsilon_{lo}^{i+1} + \frac{\epsilon_{up}^{i+1} - \epsilon_{lo}^{i+1}}{2}$ and thus the statement holds. □

**Theorem B.** Algorithm 2 terminates after $\left\lceil \frac{\log(\tau)}{\log(1/2)} \right\rceil$ iterations.

*Proof.* Algorithm 2 terminates once $\epsilon_{up} - \epsilon_{lo} \leq \tau$. By Lemma A this is equivalent to $(\frac{1}{2})^i \leq \tau$, where $i$ is the number of iterations.

$$\begin{aligned} & (\tfrac{1}{2})^i \leq \tau \\ \equiv \quad & \exp(\log(1/2) \cdot i) \leq \tau \\ \equiv \quad & \log(1/2) \cdot i \leq \log(\tau) \\ \equiv \quad & i \geq \tfrac{\log(\tau)}{\log(1/2)} \end{aligned}$$

□

**Correctness.** We first establish some lemmas that we use to prove correctness.

**Lemma C.** If there exists a non-empty $\epsilon$-relevant variable set $\mathcal{V}_{rel}$, then any $\epsilon$-relevant variable set $\mathcal{V}_{rel}'$ is non-empty.

*Proof.* By contradiction. Suppose $\mathcal{V}_{rel}' = \emptyset$ is $\epsilon$-relevant. Then $\mathcal{V}_{irr}' = \mathcal{V}$ is $\epsilon$-irrelevant. This is a contradiction, since $\mathcal{V}_{irr} = \mathcal{V} \setminus \mathcal{V}_{rel} \subset \mathcal{V}$ is $\epsilon$-irrelevant and maximal. □

Recall that in general the traversing order can influence the size of the computed $\epsilon$-relevant variable set. The lemma above guarantees that whenever there exists a non-empty $\epsilon$-relevant variable set, a non-empty $\epsilon$-relevant variable set is computed by Algorithm 1, irrespective of the traversing order.

**Lemma D.** Let $\epsilon' < \epsilon$. If there exists a non-empty $\epsilon'$-relevant variable set, then there exists a non-empty $\epsilon$-relevant variable set.

*Proof.* Let $\mathcal{V}'_{rel} \neq \emptyset$ be $\epsilon'$-relevant. Moreover, let $\mathcal{V}'_{irr} = \mathcal{V} \setminus \mathcal{V}'_{rel}$. By $\epsilon'$-relevance (Definition 1), for any $v' \in \mathcal{V}'_{rel}$ there exists $\hat{s} \in \mathcal{S}$ with $\hat{s}(v) = s_w(v)$ for each $v \in \mathcal{V}_{rel} \setminus \{v'\}$ and $\hat{s}(v) \in D_v^{\epsilon'}(s_w, s)$ for each $v \in \mathcal{V}'_{irr} \cup \{v'\}$ such that $\pi(\hat{s}) \neq l_w$. Since $\epsilon' < \epsilon$, $\hat{s}(v) \in D_v^{\epsilon'}(s_w, s)$ implies $\hat{s}(v) \in D_v^{\epsilon}(s_w, s)$ and thus there exists (non-empty) $\epsilon$-relevant $\mathcal{V}_{rel} \supseteq \mathcal{V}'_{rel}$. □

**Lemma E.** If $\epsilon' \leq \epsilon_{lo}$ (for $\epsilon_{lo}$ in Algorithm 2) then $\emptyset$ is $\epsilon'$-relevant.

*Proof.* This follows as an invariant of Algorithm 2. Initially $\epsilon_{lo} = 0$, hence $\hat{s}(v) \in D_v^{\epsilon'}(s_w, s)$ implies $\hat{s}(v) = s_w(v)$, and thus the implications holds. Moreover, whenever $\epsilon_{lo}$ is updated (line 5), we have that $\emptyset$ is $\epsilon_{lo}$-relevant. Thus, by the contrapositions of Lemma C and Lemma D, the implications holds. □

Finally, we prove **Theorem 4** (correctness BSEARCH).

*Proof.* Case distinction.

- $\mathcal{V}_{rel}$ is set (non-empty) in the loop (line 7): Whenever $\mathcal{V}_{rel}$ is updated, $\epsilon_{up}$ is set to current $\epsilon$. Hence, $\mathcal{V}_{rel}$ is $\epsilon_{up}$-relevant at loop termination. Moreover, since by Lemma E there does not exists $\epsilon' \leq \epsilon_{lo}$ with a non-empty $\epsilon'$-relevant variable set, and since $\epsilon_{up} - \epsilon_{lo} < \tau$, $\epsilon_{up}$ is minimal up to $\tau$.
- $\mathcal{V}_{rel}$ is set after loop termination (line 8): Since $\mathcal{V}_{rel} = \emptyset$ at loop termination, $\mathcal{V}_{rel}$ and thereby $\epsilon_{up}$ are not updated in the loop. Hence, it also holds $\epsilon_{up} = 1$. Moreover, since $\pi(s) \neq \pi(s_w)$, $\mathcal{V}$ is not 1-irrelevant, and thus by Lemma C a non-empty 1-relevant set $\mathcal{V}_{rel}$ is computed by VERIX$(s_w, s, 1)$ (line 8). Again, by Lemma E there does not exists $\epsilon' \leq \epsilon_{lo}$ with a non-empty $\epsilon'$-relevant variable set, and since additionally $\epsilon_{up} - \epsilon_{lo} < \tau$, 1 is minimal up to $\tau$.

□