Probabilistic Safety Verification of Neural Policies via Predicate Abstraction

Marcel Vinzent[1], **Holger Hermanns**[1], **Jörg Hoffmann**[1,2]

[1]Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
[2]German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany
{vinzent, hermanns, hoffmann}@cs.uni-saarland.de

Abstract

Neural networks are increasingly important to learn action policies. Policy predicate abstraction (PPA) verifies safety of such a neural policy π by over-approximating the state space subgraph induced by π, and using counterexample-guided abstraction refinement (CEGAR) to iteratively refine the abstraction. So far, PPA verifies safety in non-deterministic state spaces. This paper extends PPA to probabilistic verification. Extending the abstract state space computation to the probabilistic case is relatively straightforward. Abstraction refinement, however, becomes substantially more complex, due to the more intricate form of counterexamples and the various sources of spuriousness it entails. We tackle this challenge by drawing inspiration from prior work on probabilistic CEGAR, empowering it to deal with neural π. The resulting algorithm decides whether π is safe with respect to a desired upper bound on unsafety probability. Invoking the algorithm incrementally, we can also derive upper and lower bounds automatically. Our experiments show that these algorithms can derive non-trivial bounds. In a first comparison to state-of-the-art probabilistic model checkers, our approach is superior.

Technical appendix —
https://fai.cs.uni-saarland.de/vinzent/papers/ripl25-ta.pdf

Code —
https://gitlab.cs.uni-saarland.de/vinzent/PPAToolset

Introduction

Neural networks (NN) are increasingly important to learn action policies, including in AI planning and robotics (e.g., (Stahlberg et al. 2022; Amir et al. 2023). But how to verify safety of such a policy π? Given a *start condition* ϕ_0 and an *unsafety condition* ϕ_u, does there exist an unsafe state $s^u \models \phi_u$ reachable from a start state $s^0 \models \phi_0$ under π? Such verification is notoriously hard. A growing body of work addresses NN-controlled systems (Ivanov et al. 2021; Lopez et al. 2023; Wang et al. 2024), with a focus on continuous time and action spaces. Another thread explores probabilistic safety guarantees of neural policies in deterministic systems with discrete time and action spaces (Bacci and Parker 2020; Katz et al. 2023).

Here we follow up on work on *policy predicate abstraction* (PPA) by Vinzent et al. (2022; 2023; 2024) (henceforth: *Vea*), which tackles safety verification of neural action policies π in non-deterministic systems with discrete time and action spaces. General *predicate abstraction* (PA) (Graf and Saïdi 1997) builds an abstraction of the full state space Θ defined through a set $\mathcal{P}$ of *predicates* (linear constraints over the state variables, e.g., $x + y \leq 1$), over-approximating all possible behaviors. In contrast, PPA abstracts the subgraph Θ^π induced by π, over-approximating all possible behaviors under π. If the abstraction $\Theta^\pi_\mathcal{P}$ is safe, then so is π. *Counterexample-guided abstraction refinement* (CEGAR) (Clarke et al. 2003) is used to iteratively refine $\mathcal{P}$ until either $\Theta^\pi_\mathcal{P}$ is safe, or a *realizable* abstract counterexample is found, proving π unsafe.

So far, PPA supports *non-determinism* under π: multiple outgoing transitions in state s, induced by the same non-deterministic action $\pi(s)$. Here, we extend PPA to handle *probabilistic transitions* in addition to non-determinism: each (non-deterministic) transition specifies a probability distribution over the possible successor states. In this setting, π is unsafe if there exist an *adversary* A^π, a (worst-case) resolution of transition non-determinism in Θ^π, and a start state $s^0 \models \phi_0$ such that under A^π the *unsafety probability* of reaching ϕ_u from s^0 exceeds a desired upper bound p_u. Extending the abstract state space computation to the probabilistic case is relatively straightforward. Abstraction refinement, however, becomes substantially more complex. In the non-probabilistic context, an abstract counterexample is an individual path $\sigma_\mathcal{P}$ from ϕ_0 to ϕ_u in $\Theta^\pi_\mathcal{P}$. $\sigma_\mathcal{P}$ is realizable if there exists a corresponding concrete path in Θ^π. In the probabilistic context, however, an abstract counterexample is an adversary $A^\pi_\mathcal{P}$ for $\Theta^\pi_\mathcal{P}$ and an abstract start state $s^0_\mathcal{P} \models \phi_0$ such that under $A^\pi_\mathcal{P}$ the probability of reaching ϕ_u from $s^0_\mathcal{P}$ exceeds p_u. How to effectively check realizability of such counterexamples, and how to refine the abstraction if the counterexample is *spurious* (not realizable)?

We address this challenge by extending PPA with machinery inspired by prior work on probabilistic CEGAR (*prob-CEGAR*) (Hermanns et al. 2008). prob-CEGAR checks realizability by incrementally constructing a set $ups_\mathcal{P}$ of abstract unsafe paths induced by the abstract counterexample such that the probability $Pr(ups_\mathcal{P})$ exceeds p_u. There are two *sources of spuriousness*: (i) individual paths $\sigma_\mathcal{P} \in ups_\mathcal{P}$ may not be realizable, (ii) the (maximal) *realizable probability* $maxPr[ups_\mathcal{P}]$ may not exceed p_u. Here, we extend prob-CEGAR to a new CEGAR method for probabilistic PPA

(*prob-CEGAR-PPA*), handling the additional sources of spuriousness induced by the neural policy π: it may happen that (i) and (ii) do not occur in the full system Θ, but in Θ^π which restricts actions to those taken by π. The resulting algorithm decides whether π is safe with respect to a given p_u (*prob-CEGAR-PPA-p_u*). By invoking the algorithm incrementally, iterating over different p_u while re-using abstraction predicates, we can automate the derivation of upper and lower bounds on unsafety probability (*prob-CEGAR-PPA-inc*).

In our experiments these algorithms can derive non-trivial bounds. We also show that handling the probabilistic transitions in abstract state space computation incurs little overhead, and so does verifying a given p_u with prob-CEGAR-PPA-inc compared to prob-CEGAR-PPA-p_u. Finally, we run a first comparison to state-of-the-art probabilistic model checkers, via straightforward encodings of the π-controlled system into their input languages. Our approach is superior: the model checkers barely solve anything.

Overview: After providing preliminaries, we summarize prior work on probabilistic PA, followed by our extension to probabilistic PPA; and similarly for prior CEGAR methods followed by our extensions. We describe our experiments and conclude. A technical appendix (TA) is available online.

Preliminaries

$Dist(X)$ denotes the set of *probability distributions* over X, i.e., for $\mu \in Dist(X)$ it holds $\mu\colon X \to [0,1]$ and $\sum_{x\in X} \mu(x) = 1$. Throughout this work, we consider distributions with a **finite** *support* $Supp(\mu) = \{x \in X \mid \mu(x) > 0\}$.

We consider state spaces 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 interval. $\mathcal{L}$ is a finite set of *action labels*. $\mathcal{O}$ is a finite set of *action operators*. We denote by $C(\mathcal{V})$, short C, the set of *linear constraints* over $\mathcal{V}$, i.e., of the form $\sum_{v\in\mathcal{V}} d_v \cdot v \geq c$ with coefficients d_v and c, and Boolean combinations thereof. Accordingly, Exp denotes the set of *linear expressions* over $\mathcal{V}$, i.e., of the form $\sum_{v\in\mathcal{V}} d_v \cdot v + c$. An *action operator* $o \in \mathcal{O}$ is a tuple $(l, g, \bar{u})$ with *label* $l \in \mathcal{L}$, *guard* $g \in C$, and *probabilistic update* $\bar{u} \in Dist(\mathcal{V} \to Exp)$. That is, $\bar{u}(u)$ is the probability of update $u \in Supp(\bar{u})$, and u assigns each $v \in \mathcal{V}$ an update expression from Exp. We write $l(o), g(o)$ and $\bar{u}(o)$ to denote the respective structure.

The state space of $\langle \mathcal{V}, \mathcal{L}, \mathcal{O}\rangle$ is the ***probabilistic transition systems*** (PTS) $\Theta = \langle \mathcal{S}, \mathcal{O}, U, \mathcal{T}\rangle$. The set of *states* $\mathcal{S}$ is the set of *variable assignments*, i.e., $s \in \mathcal{S}$ is a function with domain $dom(s) = \mathcal{V}$ and $s(v) \in D(v)$ for each v. We denote by $\phi(s)$ and $e(s)$ the evaluation of $\phi \in C$ and $e \in Exp$ over s. We write $s \models \phi$ if $\phi(s)$ evaluates true and denote $[\phi] = \{s \in \mathcal{S} \mid s \models \phi\}$. $U = \bigcup_{o\in\mathcal{O}} Supp(\bar{u}(o))$ is the set of *updates*. $s[\![u]\!] = \{v \mapsto u(v)(s) \mid v \in \mathcal{V}\}$ denotes the evaluation of $u \in U$ over $s \in \mathcal{S}$. Accordingly, $s[\![\bar{u}]\!] \in Dist(U \times \mathcal{S})$ denotes the *U-annotated state distribution* induced by probabilistic update $\bar{u}$ in s, i.e., $s[\![\bar{u}]\!](u, s') = \bar{u}(u)$ for $u \in Supp(\bar{u})$ and $s' = s[\![u]\!]$, otherwise $s[\![\bar{u}]\!](u, s') = 0$. The set of *transitions* $\mathcal{T}$ contains $(s, o, \mu) \in \mathcal{S} \times \mathcal{O} \times Dist(U \times \mathcal{S})$ for operator $o = (l, g, \bar{u})$ iff $s \models g$ and $\mu = s[\![\bar{u}]\!]$. A *path* in Θ is a finite sequence $\langle s^0, o^0, \mu^0, u^0, \ldots, s^n\rangle$ s.t. $(s^i, o^i, \mu^i) \in \mathcal{T}$ and $(u^i, s^{i+1}) \in Supp(\mu^i)$ for all i. By $Path(\Theta)$, short $Path$, we denote the set of paths. By $s^i(\sigma)$, $o^i(\sigma)$, $\mu^i(\sigma)$ and $u^i(\sigma)$ we denote the i-th element of $\sigma \in Path$.

An ***action policy*** π is a function $\mathcal{S} \to \mathcal{L}$. The *policy-restricted* subgraph is the PTS $\Theta^\pi = \langle \mathcal{S}, \mathcal{O}, U, \mathcal{T}^\pi\rangle$ with $\mathcal{T}^\pi = \{(s, o, \mu) \in \mathcal{T} \mid \pi(s) = l(o)\}$. The distinction between action operators $\mathcal{O}$ and labels $\mathcal{L}$ (selected by π) allows for *non-determinism* in Θ^π: multiple transitions in state $s \in \mathcal{S}$, induced by multiple operators with the same label $l \in \mathcal{L}$. We consider π represented by a *neural network* (NN). We focus on feed-forward NN with *ReLU* activation $ReLU(x) = \max(x, 0)$. These NN consist of an input layer, arbitrarily many hidden layers, and an output layer with one neuron per label $l \in \mathcal{L}$.

A ***safety property*** is a tuple $\rho = (\phi_0, \phi_u, p_u)$. $\phi_0 \in C$ represents the set of *start states* and $\phi_u \in C$ the set of *unsafe states*; $p_u \in [0, 1]$ is an *upper bound on unsafety probability*. $Path(\Theta, \phi_u) = \{\sigma \in Path(\Theta) \mid s^{|\sigma|}(\sigma) \models \phi_u \wedge \forall 0 \leq i < |\sigma| : s^i(\sigma) \not\models \phi_u\}$ denotes the set of (minimal) *unsafe paths*. An ***adversary*** $A\colon \mathcal{S} \to (\mathcal{O} \times Dist(U \times \mathcal{S}))$ resolves transition non-determinism in Θ. It induces a probability measure on the *Markov chain* $\Theta^A = \langle \mathcal{S}, \mathcal{O}, U, \mathcal{T}^A\rangle$ where $\mathcal{T}^A = \{(s, o, \mu) \in \mathcal{T} \mid A(s) = (o, \mu)\}$. $Path(\Theta^A, s, \phi_u) = \{\sigma \in Path(\Theta^A, \phi_u) \mid s^0(\sigma) = s\}$ denotes the unsafe paths under A from $s \in \mathcal{S}$, short $Path(A, s, \phi_u)$. $Pr(ups) = \sum_{\sigma\in ups} \prod_{i=0}^{|\sigma|-1} (\mu^i(u^i, s^{i+1}))(\sigma)$ is the accumulated *unsafety probability* of $ups \subseteq Path(A, s, \phi_u)$. We abbreviate $Pr(\sigma) = Pr(\{\sigma\})$ and $Pr(A, s, \phi_u) = Pr(Path(A, s, \phi_u))$. Θ is *unsafe* with respect to ρ iff there exists a *counterexample* (A, s^0), i.e., an adversary A and a start state $s^0 \models \phi_0$ such that $Pr(A, s^0, \phi_u) > p_u$. Otherwise Θ is *safe*. Safety straight-forwardly translates to **policy safety**: π is safe iff Θ^π is safe.

Prior Work: Probabilistic PA

Probabilistic predicate abstraction (Wachter et al. 2007) adopts predicate abstraction (Graf and Saïdi 1997) for probabilistic systems. 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 *concretization* of $s_\mathcal{P}$. The abstraction of distribution $\mu \in Dist(U \times \mathcal{S})$ is the distribution $\mu|_\mathcal{P} = \{(u, s_\mathcal{P}) \mapsto \sum_{s\in[s_\mathcal{P}]} \mu(u, s) \mid u \in U, s_\mathcal{P} \in \mathcal{S}_\mathcal{P}\}$ and $[\mu_\mathcal{P}] = \{\mu \in Dist(U \times \mathcal{S}) \mid \mu|_\mathcal{P} = \mu_\mathcal{P}\}$ denotes the concretization of $\mu_\mathcal{P} \in Dist(U \times \mathcal{S}_\mathcal{P})$.

Definition 1 (Probabilistic Predicate Abstraction)**.** The *predicate abstraction* of Θ over $\mathcal{P}$ is the PTS $\Theta_\mathcal{P} = \langle \mathcal{S}_\mathcal{P}, \mathcal{O}, U, \mathcal{T}_\mathcal{P}\rangle$ where $\mathcal{S}_\mathcal{P}$ is the set of all abstract states over $\mathcal{P}$ and $\mathcal{T}_\mathcal{P} = \{(s_\mathcal{P}, o, \mu_\mathcal{P}) \in \mathcal{S}_\mathcal{P} \times \mathcal{O} \times Dist(U \times \mathcal{S}_\mathcal{P}) \mid \exists s \in [s_\mathcal{P}], \mu \in [\mu_\mathcal{P}]\colon (s, o, \mu) \in \mathcal{T}\}$.

To compute $\Theta_\mathcal{P}$, one must solve the *abstract transition problem* for every possible abstract transition: $(s_\mathcal{P}, o, \mu_\mathcal{P}) \in$

$\mathcal{T}_\mathcal{P}$ with $o = (l, g, \bar{u})$ iff there exist a concrete state $s \in [s_\mathcal{P}]$ such that $s \models g(o)$ and $s[\![\bar{u}]\!] \in [\mu_\mathcal{P}]$. This is routinely encoded into satisfiability modulo theories (SMT) using off-the-shelf solvers (de Moura and Bjørner 2008).

The abstraction of a concrete path $\sigma \in Path(\Theta)$ is the abstract path $\sigma|_\mathcal{P} \in Path(\Theta_\mathcal{P})$ where $s^i(\sigma|_\mathcal{P}) = s^i(\sigma)|_\mathcal{P}, o^i(\sigma|_\mathcal{P}) = o^i(\sigma), \mu^i(\sigma|_\mathcal{P}) = \mu^i(\sigma)|_\mathcal{P}$, and $u^i(\sigma|_\mathcal{P}) = u^i(\sigma)$ for all $i \in 0, \ldots, |\sigma|$. Conversely, $[\sigma_\mathcal{P}](\Theta) = \{\sigma \in Path(\Theta) \mid \sigma|_\mathcal{P} = \sigma_\mathcal{P}\}$, short $[\sigma_\mathcal{P}]$, denotes the concretization of abstract path $\sigma_\mathcal{P} \in Path(\Theta_\mathcal{P})$. The concretization of an abstract adversary $A_\mathcal{P}$ is the concrete adversary $[A_\mathcal{P}]$: $[A_\mathcal{P}](s) = (o, s[\![\bar{u}(o)]\!])$ if $A_\mathcal{P}(s|_\mathcal{P}) = (o, \mu_\mathcal{P})$ and $s[\![\bar{u}(o)]\!] \in [\mu_\mathcal{P}]$. Otherwise $[A_\mathcal{P}](s) = (o^\delta, \mu^\delta)$, where (o^δ, μ^δ) is a special element s.t. $(s, o^\delta, \mu^\delta) \notin \mathcal{T}$, i.e., $[A_\mathcal{P}]$ *blocks*.

For $\phi \in C$ let $s_\mathcal{P} \models \phi$ iff $\exists s \in [s_\mathcal{P}] \cap [\phi]$. Safety extends straight-forwardly to $\Theta_\mathcal{P}$. Due to the over-approximating, safety of $\Theta_\mathcal{P}$ implies safety of Θ. Proofs for all theoretical results are shown in the appendix.

Proposition 2 (Safety in $\Theta_\mathcal{P}$)**.** *Let $\rho = (\phi_0, \phi_u, p_u)$. If $\Theta_\mathcal{P}$ is safe with respect to ρ, then Θ is safe as well.*

Probabilistic PPA

Policy predicate abstraction (PPA) (Vea 2022) does not abstract the full state space Θ, but rather the subgraph Θ^π induced by neural policy π. Our contribution here is the extension of PPA to probabilistic verification. In this section, we introduce the probabilistic PPA state space $\Theta^\pi_\mathcal{P}$, the computation of $\Theta^\pi_\mathcal{P}$ and of abstract counterexamples. Afterward we show how to obtain the predicate set $\mathcal{P}$.

Definition 3 (Policy Predicate Abstraction)**.** The *policy predicate abstraction* (PPA) of Θ over $\mathcal{P}$ and π is the PTS $\Theta^\pi_\mathcal{P} = \langle \mathcal{S}_\mathcal{P}, \mathcal{O}, U, \mathcal{T}^\pi_\mathcal{P} \rangle$ where $\mathcal{T}^\pi_\mathcal{P} = \{(s_\mathcal{P}, o, \mu_\mathcal{P}) \in \mathcal{S}_\mathcal{P} \times \mathcal{O} \times Dist(U \times \mathcal{S}_\mathcal{P}) \mid \exists s \in [s_\mathcal{P}], \mu \in [\mu_\mathcal{P}] \colon (s, o, \mu) \in \mathcal{T}^\pi\}$.

Safety of π can be proven via safety of $\Theta^\pi_\mathcal{P}$.

Proposition 4 (Safety in $\Theta^\pi_\mathcal{P}$)**.** *Let $\rho = (\phi_0, \phi_u, p_u)$. If $\Theta^\pi_\mathcal{P}$ is safe with respect to ρ, then π is safe as well.*

Computing the abstract state space. The *abstract transition problem* for $\Theta^\pi_\mathcal{P}$ extends the one for $\Theta_\mathcal{P}$ in that $(s_\mathcal{P}, o, \mu_\mathcal{P}) \in \mathcal{T}^\pi_\mathcal{P}$ with $o = (l, g, \bar{u})$ iff there exist a concrete state $s \in [s_\mathcal{P}]$ such that $s \models g(o)$, $s[\![\bar{u}]\!] \in [\mu_\mathcal{P}]$, and **additionally** $\pi(s) = l(o)$, i.e., π selects $l(o)$ in s. This is a key source of intricacy since the SMT formula representing the neural policy π contains one non-linear constraint per ReLU activation. This also pertains to non-probabilistic PPA. Yet, the encoding for probabilistic PPA is again more laborious since it involves a successor distribution (essentially a conjunction of abstract states) rather than a single successor. Our SMT encodings are shown in the TA.

Vea (2022) introduce an algorithm to efficiently compute $\Theta^\pi_\mathcal{P}$ for non-probabilistic systems, more specifically for the expansion of an abstract source state $s_\mathcal{P}$. They deploy approximate SMT checks embedded into an exact decision procedure. In particular continuous relaxation of integer state variable enables to use NN-tailored SMT solvers (Katz et al. 2019). These enhancements can be directly lifted to probabilistic PPA. Probabilistic expansion is more complex in that it must enumerate abstract successor distributions while non-probabilistic expansion enumerates abstract successor states. That said, the size of distribution $\mu_\mathcal{P}$ is bounded by the size of probabilistic update $\bar{u}$, $|Supp(\mu_\mathcal{P})| = |Supp(\bar{u})|$, and hence usually small in practice. Pseudocode adapted for probabilistic expansion is available in the TA.

Computing abstract counterexamples. To verify safety in $\Theta^\pi_\mathcal{P}$, we must check for a counterexample $(A^\pi_\mathcal{P}, s^0_\mathcal{P})$. This boils down to a MaxProb problem (Steinmetz et al. 2016): $\Theta^\pi_\mathcal{P}$ is unsafe iff $maxPr(\Theta^\pi_\mathcal{P}, \phi_0, \phi_u) > p_u$ where $maxPr(\Theta^\pi_\mathcal{P}, \phi_0, \phi_u) = \max_{A^\pi_\mathcal{P}, s^0_\mathcal{P} \models \phi_0} Pr(A^\pi_\mathcal{P}, s^0_\mathcal{P}, \phi_u)$ is the maximal unsafety probability of $\Theta^\pi_\mathcal{P}$. This problem can be solved with value iteration (Bertsekas 1995), but that necessitates to construct the entire reachable fragment of $\Theta^\pi_\mathcal{P}$. We hence use *FRET-LRTDP* (Steinmetz et al. 2016), which explores the reachable fragment incrementally and terminates once maximizing $(A^\pi_\mathcal{P}, s^0_\mathcal{P})$ is computed. $A^\pi_\mathcal{P}$ may be *partial but closed* in that it is defined on the fragment of $\Theta^\pi_\mathcal{P}$ that is reachable from $s^0_\mathcal{P}$ under $A^\pi_\mathcal{P}$ while it may be undefined elsewhere. This enables constructing the reachable fragment only partially, while still solving MaxProb exactly.

Prior Work: CEGAR

So far we have not specified how to derive the predicate set $\mathcal{P}$. The commonly used method in PA is *counterexample-guided abstraction refinement* (CEGAR) (Clarke et al. 2003). Starting from simple $\mathcal{P}$ – here $\mathcal{P} = \{\phi_u\}$ sufficient to distinguish unsafe states – $\mathcal{P}$ is iteratively refined. If $\Theta_\mathcal{P}$ is safe, then so is the concrete state space Θ and CEGAR stops, else there exists an abstract counterexample in $\Theta_\mathcal{P}$. If the counterexample is *realizable* in Θ, then Θ is unsafe and CEGAR stops, else the counterexample is *spurious* and new predicates are added to $\mathcal{P}$ to eliminate spuriousness. Specific methods are required to deal with the possible sources of spuriousness in (1) PPA; and (2) probabilistic state spaces. Here, we summarize prior work tackling (1) and (2). In the next section, we combine the two in order to obtain a CEGAR method for probabilistic PPA.

CEGAR for non-probabilistic PPA. Vea (2023) provide a CEGAR framework specialized to non-probabilistic PPA (CEGAR-PPA). Here, a counterexample is an abstract unsafe path $\sigma_\mathcal{P}$ from ϕ_0 to ϕ_u, found via forward search in $\Theta^\pi_\mathcal{P}$. Let $[\sigma_\mathcal{P} \wedge \phi_u](\Theta) = [\sigma_\mathcal{P}](\Theta) \cap Path(\Theta, \phi_u)$ denote the set of unsafe concretization paths of $\sigma_\mathcal{P}$ in Θ. $\sigma_\mathcal{P}$ is realizable iff there exists $\sigma \in [\sigma_\mathcal{P} \wedge \phi_u](\Theta^\pi)$ with $s^0(\sigma) \models \phi_0$. If $\sigma_\mathcal{P}$ is spurious, $\mathcal{P}$ is refined by adding predicates based on the source of spuriousness.

(a) **$\mathcal{T}$-spuriousness** is induced by the transition behavior $\mathcal{T}$ of the system. $\sigma_\mathcal{P}$ is *$\mathcal{T}$-realizable* iff there exists a *non-policy-restricted* concretization $\sigma \in [\sigma_\mathcal{P} \wedge \phi_u](\Theta)$ with $s^0(\sigma) \models \phi_0$. Otherwise, $\sigma_\mathcal{P}$ is *$\mathcal{T}$-spurious*. $\mathcal{T}$-spuriousness also occurs in general non-policy-restricted systems. The spuriousness check is routinely encoded into SMT. Standard refinement techniques exist. Vea deploy weakest precondition computation along $\sigma_\mathcal{P}$.

(b) **π-spuriousness** is induced by the policy π. A $\mathcal{T}$-realizable abstract path $\sigma_{\mathcal{P}}$ is *π-realizable* iff there exists a *policy-restricted* concretization $\sigma \in [\sigma_{\mathcal{P}} \wedge \phi_u](\Theta^\pi)$ with $s^0(\sigma) \models \phi_0$. Otherwise, $\sigma_{\mathcal{P}}$ is *π-spurious*. The spuriousness check can be encoded in SMT. For refinement Vea introduce *witness splitting*. Consider $s^i_{\mathcal{P}}(\sigma_{\mathcal{P}})$ for some $i < |\sigma_{\mathcal{P}}|$. $\sigma_{\mathcal{P}}$ proceeds due to some concrete witness $(s^i_w, o^i(\sigma_{\mathcal{P}}), s^{i+1}_w) \in \mathcal{T}^\pi$ with $s^i_w \in [s^i_{\mathcal{P}}(\sigma_{\mathcal{P}})]$ and $s^{i+1}_w \in [s^{i+1}_{\mathcal{P}}(\sigma_{\mathcal{P}})]$. $\sigma_{\mathcal{P}}$ is π-spurious if no such s^i_w is reachable via (prefix) concretizations of $\sigma_{\mathcal{P}}$ in Θ^π. In other words, $\pi(s^i(\sigma)) \neq l(o^i(\sigma_{\mathcal{P}}))$ for any concretization $\sigma \in [\sigma_{\mathcal{P}} \wedge \phi_u](\Theta)$. Vea introduce split predicates to distinguish s^i_w and $s^i(\sigma)$ in the refined abstraction, approximating the decision boundary of π in $s^i_{\mathcal{P}}(\sigma_{\mathcal{P}})$.

CEGAR for probabilistic PA. *Probabilistic CEGAR* (prob-CEGAR) (Hermanns et al. 2008) extends CEGAR to probabilistic systems. Let $(A_{\mathcal{P}}, s^0_{\mathcal{P}})$ be an abstract counterexample, i.e., $s^0_{\mathcal{P}} \models \phi_0$ and $Pr(A_{\mathcal{P}}, s^0_{\mathcal{P}}, \phi_u) > p_u$. $(A_{\mathcal{P}}, s^0_{\mathcal{P}})$ is realizable iff there exists some start state $s^0 \in [s^0_{\mathcal{P}}] \cap [\phi_0]$ such that $Pr([A_{\mathcal{P}}], s^0, \phi_u) > p_u$. To check this, prob-CEGAR maintains a finite set of abstract unsafe paths $ups_{\mathcal{P}} \subseteq Path(A_{\mathcal{P}}, s^0_{\mathcal{P}}, \phi_u)$. Individual paths $\sigma_{\mathcal{P}}$ are added to $ups_{\mathcal{P}}$ incrementally with decreasing probability $Pr(\sigma_{\mathcal{P}})$ via weighted search in ${\Theta_{\mathcal{P}}}^{A_{\mathcal{P}}}$ (Han and Katoen 2007), exploiting that iff $Pr(A_{\mathcal{P}}, s^0_{\mathcal{P}}, \phi_u) > p_u$ then there exists finite $ups_{\mathcal{P}}$ such that $Pr(ups_{\mathcal{P}}) > p_u$ (Han et al. 2009).

There are two possible **sources of spuriousness**. (i) **Individual paths** $\sigma_{\mathcal{P}}$ may be spurious. This corresponds to non-probabilistic spuriousness (a) and is checked and refined using standard techniques. (ii) For $Pr(ups_{\mathcal{P}}) > p_u$, the **maximal realizable probability** of $ups_{\mathcal{P}}$ may not exceed p_u because concretizations of distinct abstract paths in $ups_{\mathcal{P}}$ may have distinct start states. Let $ups_{\mathcal{P}}|_s = \{\sigma_{\mathcal{P}} \in ups_{\mathcal{P}} \mid \exists \sigma \in [\sigma_{\mathcal{P}} \wedge \phi_u] \colon s^0(\sigma) = s\}$ denote the subset of $ups_{\mathcal{P}}$ with an unsafe concretization from $s \in \mathcal{S}$. The maximal realizable probability of $ups_{\mathcal{P}}$ is $maxPr[ups_{\mathcal{P}}] = \max_{s^0 \in [s^0_{\mathcal{P}}] \cap [\phi_0]} Pr(ups_{\mathcal{P}}|_{s^0})$. $maxPr[ups_{\mathcal{P}}]$ can be computed via an encoding into MaxSMT (Bjørner and Phan 2014). If $maxPr[ups_{\mathcal{P}}] > p_u$ then $ups_{\mathcal{P}}$ and thereby the abstract counterexample $(A_{\mathcal{P}}, s^0_{\mathcal{P}})$ are realizable. If $maxPr[ups_{\mathcal{P}}] \leq p_u$ then $(A_{\mathcal{P}}, s^0_{\mathcal{P}})$ is not necessarily spurious. $ups_{\mathcal{P}}$ is only a subset of $Path(A_{\mathcal{P}}, s^0_{\mathcal{P}}, \phi_u)$. $P_{unused} = Pr(A_{\mathcal{P}}, s^0_{\mathcal{P}}, \phi_u) - Pr(ups_{\mathcal{P}})$ is the "unused" probability mass of $(A_{\mathcal{P}}, s^0_{\mathcal{P}})$. $(A_{\mathcal{P}}, s^0_{\mathcal{P}})$ is spurious if $maxPr[ups_{\mathcal{P}}] + P_{unused} \leq p_u$. Refinement introduces predicates to distinguish start states in $s^0_{\mathcal{P}}$.

CEGAR for Probabilistic PPA

We contribute a CEGAR method for probabilistic policy verification via PPA (prob-CEGAR-PPA). It combines ideas from CEGAR-PPA and prob-CEGAR (cf. previous section). Algorithm 1 shows pseudocode. We first discuss the sources of spuriousness and their refinement; then we describe and formally analyze our algorithm. Afterward, we provide an extension of prob-CEGAR-PPA for the automated derivation of upper and lower bounds on unsafety probability.

Algorithm 1: Probabilistic CEGAR for PPA.

Input: $\langle \mathcal{V}, \mathcal{L}, \mathcal{O} \rangle, (\phi_0, \phi_u, p_u), p_\varepsilon > 0$ (minimal path probability).

$\mathcal{P} \leftarrow \{\phi_u\}$
while 1 **do**
  $(A^\pi_{\mathcal{P}}, s^0_{\mathcal{P}}) \leftarrow$ `max-prob`$(\langle \mathcal{V}, \mathcal{L}, \mathcal{O} \rangle, (\phi_0, \phi_u, p_u), \mathcal{P}, \pi)$
  if $Pr(A^\pi_{\mathcal{P}}, s^0_{\mathcal{P}}, \phi_u) \leq p_u$ **then return** *SAFE*
  if `CeAna`$(A^\pi_{\mathcal{P}}, s^0_{\mathcal{P}})$ $=$ *REAL* **then return** *UNSAFE*
Procedure `CeAna`$(A^\pi_{\mathcal{P}}, s^0_{\mathcal{P}})$**:**
  $ups_{\mathcal{P}} \leftarrow \emptyset$
  $P_{max} \leftarrow 0$ `// Upper bound on` $maxPr[ups_{\mathcal{P}}]$.
  $P_{unused} = Pr(A^\pi_{\mathcal{P}}, s^0_{\mathcal{P}}, \phi_u) - Pr(ups_{\mathcal{P}})$ `// Notation.`
  can-add$(ups_{\mathcal{P}}) := Pr(ups_{\mathcal{P}}) < p_u \vee p_\varepsilon < \min\{Pr(\sigma_{\mathcal{P}}) \mid \sigma_{\mathcal{P}} \in ups_{\mathcal{P}}\}$ `// Termination condition.`
  while 1 **do**
    while $P_{max} \leq p_u \wedge$ *can-add*$(ups_{\mathcal{P}})$ **do**
      $\sigma_{\mathcal{P}} \leftarrow (|ups_{\mathcal{P}}| + 1)$`-HighestProb`$(A^\pi_{\mathcal{P}}, s^0_{\mathcal{P}}, \phi_u)$
      if $\neg$*is-*$\mathcal{T}$*-path-realizable*$(\sigma_{\mathcal{P}})$ **then**
        return *refine-*$\mathcal{T}$*-path*$(\mathcal{P}, \sigma_{\mathcal{P}})$ `// (i-a)`
      $ups_{\mathcal{P}} \leftarrow ups_{\mathcal{P}} \cup \{\sigma_{\mathcal{P}}\}$
      $P_{max} \leftarrow P_{max} + Pr(\sigma_{\mathcal{P}})$
    $P_{max} \leftarrow maxPr[ups_{\mathcal{P}}](\Theta)$ `// MaxSMT`
    if $P_{max} > p_u$ **then break**
    if $P_{max} + P_{unused} \leq p_u \vee \neg$*can-add*$(ups_{\mathcal{P}})$ **then**
      return *refine-*$\mathcal{T}$*-prob*$(\mathcal{P}, ups_{\mathcal{P}})$ `// (ii-a)`
  let $ups^*_{\mathcal{P}} \subseteq ups_{\mathcal{P}}$ **with** $maxPr[ups^*_{\mathcal{P}}](\Theta) = Pr(ups^*_{\mathcal{P}}) = P_{max}$
  if $\exists \sigma_{\mathcal{P}} \in ups^*_{\mathcal{P}} \colon \neg$*is-*$\pi$*-path-realizable*$(\sigma_{\mathcal{P}})$ **then**
    return *refine-*π*-path*$(\mathcal{P}, \sigma_{\mathcal{P}})$ `// (i-b)`
  if $\neg$*is-*π*-all-realizable*$(ups^*_{\mathcal{P}})$ **then**
    return *refine-*π*-prob*$(\mathcal{P}, ups^*_{\mathcal{P}})$ `// (ii-b)`
  return *REAL*

Spuriousness and refinement. Consider an abstract counterexample $(A^\pi_{\mathcal{P}}, s^0_{\mathcal{P}})$ in $\Theta^\pi_{\mathcal{P}}$. prob-CEGAR-PPA composes the sources of spuriousness encountered in CEGAR-PPA and prob-CEGAR. It involves two sources of spuriousness of individual paths, specifically **(i-a) $\mathcal{T}$-path** and **(i-b) π-path spuriousness**. We check and refine both sources analogously to non-probabilistic CEGAR-PPA (Vea 2024).

As in prob-CEGAR, prob-CEGAR-PPA maintains $ups_{\mathcal{P}} \subseteq Path(A^\pi_{\mathcal{P}}, s^0_{\mathcal{P}}, \phi_u)$, where each $\sigma_{\mathcal{P}} \in ups_{\mathcal{P}}$ is $\mathcal{T}$-path realizable. There are two sources of *probabilistic spuriousness*. **(ii-a) $\mathcal{T}$-probabilistic spuriousness** is induced by the transition behavior $\mathcal{T}$. It corresponds to source (ii) of non-policy-restricted prob-CEGAR. $ups_{\mathcal{P}}$ is *$\mathcal{T}$-probabilistic realizable* iff it has sufficient realizable probability *without policy-restriction*, $maxPr[ups_{\mathcal{P}}](\Theta) > p_u$. Otherwise it is *$\mathcal{T}$-probabilistic spurious*. We compute $maxPr[ups_{\mathcal{P}}](\Theta)$ via a call to MaxSMT.

(ii-b) π-probabilistic spuriousness is induced by the policy π. This new source is specific to prob-CEGAR-PPA. Consider $\mathcal{T}$-probabilistic realizable $ups_{\mathcal{P}}$ such that each $\sigma_{\mathcal{P}} \in ups_{\mathcal{P}}$ is π-path realizable. $ups_{\mathcal{P}}$ is *π-probabilistic realizable* iff it has sufficient realizable probability *under* π, $maxPr[ups_{\mathcal{P}}](\Theta^\pi) > p_u$. Otherwise it is *$\pi$-probabilistic spurious*. Checking (ii-b) is non-trivial. $maxPr[ups_{\mathcal{P}}](\Theta^\pi)$ boils down to a NN-constrained MaxSMT problem. Support

for NN-tailored MaxSMT solvers is limited (Strong et al. 2023). We settle for an *under-approximative* π-realization check: Let $ups^*_{\mathcal{P}} \subseteq ups_{\mathcal{P}}$ be a maximal $\mathcal{T}$-realizable subset, i.e., $maxPr[ups^*_{\mathcal{P}}](\Theta) = Pr(ups^*_{\mathcal{P}}) = maxPr[ups_{\mathcal{P}}](\Theta)$. $ups_{\mathcal{P}}$ is π-probabilistic realizable if $ups^*_{\mathcal{P}}$ is π-*all realizable* under π, i.e., if there exists $s^0 \in [s^0_{\mathcal{P}}] \cap [\phi_0]$ such that for all $\sigma_{\mathcal{P}} \in ups^*_{\mathcal{P}}$ there exists $\sigma \in [\sigma_{\mathcal{P}} \wedge \phi_u](\Theta^\pi)$ with $s^0(\sigma) = s^0$. This check can be encoded in NN-tailored SMT. It is under-approximative in that, even if $ups^*_{\mathcal{P}}$ is not π-all realizable, $ups_{\mathcal{P}}$ may be π-probabilistic realizable, i.e., there may exist another π-all realizable subset $ups'_{\mathcal{P}} \subseteq ups_{\mathcal{P}}$ with sufficient probability mass $Pr(ups'_{\mathcal{P}}) > p_u$.

For refinement of probabilistic spuriousness, we propose ***start witness splitting***. Consider a finite set of start states $\mathcal{S}^0 \subseteq [s^0_{\mathcal{P}}] \cap [\phi_0]$. For each state variable $v \in \mathcal{V}$ let $\mathcal{S}^0(v) = \{s(v) \mid s \in \mathcal{S}^0\}$ denote the set of start state values and let $\mathcal{S}^0_i(v) \in \mathcal{S}^0(v)$ denote the i-th smallest value. We introduce split predicates $v \leq \mathcal{S}^0_i(v)$ for each $i \in 1 \ldots, \left|\mathcal{S}^0(v)\right| - 1$. $\mathcal{S}^0$ is constructed as follows: **(ii-a)** $\mathcal{S}^0$ contains a maximizing start state $s^0_* \in [\phi_0]$ with $ups_{\mathcal{P}}|_{s^0_*} = ups^*_{\mathcal{P}}$ of some maximal $\mathcal{T}$-realizable subset $ups^*_{\mathcal{P}} \subseteq ups_{\mathcal{P}}$. Additionally, $\mathcal{S}^0$ contains a start state $s^0(\sigma)$ of some unsafe concretization $\sigma \in [\sigma_{\mathcal{P}} \wedge \phi_u](\Theta)$ for each $\sigma_{\mathcal{P}} \in ups_{\mathcal{P}} \setminus ups^*_{\mathcal{P}}$. **(ii-b)** Analogously, $\mathcal{S}^0$ contains a start state $s^0(\sigma)$ of some unsafe concretization $\sigma \in [\sigma_{\mathcal{P}} \wedge \phi_u](\Theta^\pi)$ for each $\sigma_{\mathcal{P}} \in ups^*_{\mathcal{P}}$. Intuitively, each $s^0 \in \mathcal{S}^0$ is a *witness* of the *realizable start region* of at least one $\sigma_{\mathcal{P}} \in ups_{\mathcal{P}}$. By introducing split predicates between these witnesses, we approximate the start regions in the refined abstraction.

Algorithm. The core of Algorithm 1 is the **CEGAR loop** (line 2 - 5). Each iteration starts with a MaxProb search in $\Theta^\pi_{\mathcal{P}}$ (line 3), computing the abstract adversary $A^\pi_{\mathcal{P}}$ and start state $s^0_{\mathcal{P}}$ that maximize the abstract unsafety probability $Pr(A^\pi_{\mathcal{P}}, s^0_{\mathcal{P}}, \phi_u)$. If $\Theta^\pi_{\mathcal{P}}$ and thereby π are proven safe (line 4), CEGAR terminates. Otherwise *counterexample analysis* (CeAna) is invoked (line 5). If CeAna returns *REAL*, π is unsafe and CEGAR terminates. Otherwise CEGAR iterates.

Procedure `CeAna` (line 6) shows pseudocode for **CeAna**. Following prob-CEGAR, we maintain a set of abstract unsafe paths $ups_{\mathcal{P}} \subseteq Path(A^\pi_{\mathcal{P}}, s^0_{\mathcal{P}}, \phi_u)$ (line 7). Additionally, we maintain an upper bound P_{max} on the maximal realizable probability of $ups_{\mathcal{P}}$ (line 8). P_{unused} is a short-hard notation for the abstract probability mass that is not used in $ups_{\mathcal{P}}$ (line 9). We also define a condition *can-add*$(ups_{\mathcal{P}})$ (line 10) on the minimal probability p_ε of individual paths $\sigma_{\mathcal{P}}$ added to $ups_{\mathcal{P}}$. $Pr(A^\pi_{\mathcal{P}}, s^0_{\mathcal{P}}, \phi_u)$ may be infinite. *can-add*$(ups_{\mathcal{P}})$ is to guarantee termination. The condition is suppressed while insufficient abstract probability mass has been accumulated, $Pr(ups_{\mathcal{P}}) < p_u$. $Pr(ups_{\mathcal{P}}) \geq p_u$ can always be achieved with finitely many paths (Han et al. 2009). `CeAna` first analyzes $\mathcal{T}$-spuriousness (line 11 - 21). Only if $ups_{\mathcal{P}}$ is $\mathcal{T}$-realizable, it proceeds to the more expensive analysis of π-spuriousness (line 21 - 27).

$\mathcal{T}$-analysis iteratively constructs $ups_{\mathcal{P}}$ (line 12 - 17). In each iteration, the path $\sigma_{\mathcal{P}} \in Path(A^\pi_{\mathcal{P}}, s^0_{\mathcal{P}}, \phi_u)$ with the $(|ups_{\mathcal{P}}| + 1)$-th highest probability is computed (line 13) via weighted search in the subgraph induced by $A^\pi_{\mathcal{P}}$ (Han and Katoen 2007). If $\sigma_{\mathcal{P}}$ is $\mathcal{T}$-path spurious, we refine (line 15) and `CeAna` terminates. If $\sigma_{\mathcal{P}}$ is realizable, it is added to $ups_{\mathcal{P}}$ (line 16) and we update P_{max} (line 17). Once $P_{max} > p_u$ (line 12) the maximal $\mathcal{T}$-realizable probability is computed via MaxSMT and P_{max} is set accordingly (line 18). If $P_{max} > p_u$, then $ups_{\mathcal{P}}$ is $\mathcal{T}$-realizable and we continue to π-analysis (line 19). If $P_{max} + P_{unused} \leq p_u$ (line 20), then $(A^\pi_{\mathcal{P}}, s^0_{\mathcal{P}})$ is spurious, we refine and `CeAna` terminates. We also refine if $\neg$*can-add*$(ups_{\mathcal{P}})$, i.e., the minimal probability condition on individual paths is violated. If not, $\mathcal{T}$-analysis iterates and adds additional paths to $ups_{\mathcal{P}}$ until again $P_{max} > p_u$ (or $\neg$*can-add*$(ups_{\mathcal{P}})$).

π-analysis restricts to a maximal $\mathcal{T}$-realizable subset $ups^*_{\mathcal{P}} \subseteq ups_{\mathcal{P}}$ (line 22). If some path $\sigma_{\mathcal{P}} \in ups^*_{\mathcal{P}}$ is π-path spurious, we refine (line 24) and `CeAna` terminates. Otherwise π-analysis checks whether $ups^*_{\mathcal{P}}$ is π-all realizable (line 25). If so, $(A^\pi_{\mathcal{P}}, s^0_{\mathcal{P}})$ is realizable (line 27). If not, we refine (line 26). In either case `CeAna` terminates.

Formal guarantees. CeAna involves a non-trivial loop which enumerates paths from the potentially infinitely large set $Path(A^\pi_{\mathcal{P}}, s^0_{\mathcal{P}}, \phi_u)$. This raises question of termination of CeAna in particular and prob-CEGAR-PPA in general. We answer this question in the affirmative and argue correctness.

Theorem 5. `CeAna` *terminates.*

Proof sketch. It suffices to show that $\mathcal{T}$-analysis (line 11 - 21) terminates. At the start of each iteration, it holds $P_{max} \leq p_u$ and *can-add*$(ups_{\mathcal{P}})$ (line 19, 20). Hence, the loop (line 12 - 17) is entered. Per invocation of `CeAna`, this loop can accumulate only finitely many iterations since for finitely many paths $Pr(ups_{\mathcal{P}}) > p_u$ (Han et al. 2009) and $\min_{\sigma_{\mathcal{P}} \in ups_{\mathcal{P}}} Pr(\sigma_{\mathcal{P}}) < p_\varepsilon$. Hence, $\mathcal{T}$-analysis terminates. □

Theorem 6. *If* `CeAna` *returns REAL, then* π *is unsafe.*

Proof sketch. By π-all realization of $ups^*_{\mathcal{P}}$ (line 25), there exists $ups \subseteq Path([A^\pi_{\mathcal{P}}], s^0, \phi_u)$ for some $s^0 \in [s^0_{\mathcal{P}}] \cap [\phi_0]$ so that $Pr(ups) = Pr(ups^*_{\mathcal{P}}) > p_u$. Hence, π is unsafe. □

Due to the termination condition $\neg$*can-add*$(ups_{\mathcal{P}})$, but also the under-approximative check for π-probabilistic realization (line 25), CeAna may fail to detect realizable $(A^\pi_{\mathcal{P}}, s^0_{\mathcal{P}})$. That said, for the overall CEGAR algorithm termination and correctness are guaranteed.

Theorem 7. *Algorithm 1 terminates.*

Proof sketch. In each iteration, CEGAR either terminates or strictly refines the abstraction $\mathcal{P} \subsetneq \mathcal{P}'$ in that $\exists s, t \in \mathcal{S}$ such that $s|_{\mathcal{P}} = t|_{\mathcal{P}}$ for original $\mathcal{P}$ while $s|_{\mathcal{P}'} \neq t|_{\mathcal{P}'}$ for refined $\mathcal{P}'$. This holds for refinement of (i-a,b) (Vea 2023) and also for refinement of (ii-a,b) via start state splitting, specifically for $s, t \in \mathcal{S}^0$. Since $\mathcal{S}$ is finite, $\Theta^\pi_{\mathcal{P}}$ approximates Θ^π exactly within finitely many iterations and CEGAR terminates. □

Theorem 8. *If Algorithm 1 returns SAFE, then* π *is safe. If Algorithm 1 returns UNSAFE, then* π *is unsafe.*

Proof. Follows from Proposition 4 and Theorem 6. □

Automated bound derivation. While verifying safety with respect to a given upper bound p_u, prob-CEGAR-PPA also derives interval bounds $[P_u^{lo}, P_u^{up}]$ on the maximal unsafety probability $maxPr(\Theta^\pi, \phi_0, \phi_u)$ under π. $0 \leq P_u^{lo} = maxPr[ups_\mathcal{P}](\Theta^\pi)$ is the maximal realizable probability of some $ups_\mathcal{P}$ (if computed). $P_u^{up} = maxPr(\Theta^\pi_\mathcal{P}, \phi_0, \phi_u) \leq 1$ is the (latest) maximal abstract unsafety probability of $\Theta^\pi_\mathcal{P}$.

We use linear search on p_u to incrementally tighten $[P_u^{lo}, P_u^{up}]$. The linear search is embedded into the CEGAR loop. While $\Theta^\pi_\mathcal{P}$ is safe, p_u is decreased by some step size $p_\Delta \in (0, 1)$. While CeAna derives unsafe, p_u is increased by p_Δ and CeAna re-continues with current $ups_\mathcal{P}$ and updated p_u. If spuriousness is detected, linear search pauses and CEGAR iterates. p_Δ controls the precision with which $[P_u^{lo}, P_u^{up}]$ approximates the unsafety probability in Θ^π. Linear search terminates once $P_u^{up} - P_u^{lo} \leq p_\Delta$. We denote this incremental invocation of prob-CEGAR-PPA for automated bound derivation by *prob-CEGAR-PPA-inc*. We denote prob-CEGAR-PPA verifying a specific p_u by *prob-CEGAR-PPA-p_u*. Pseudocode adapted for CEGAR with linear search is available in the TA.

Experiments

We implemented our approach on top of Vea's C++ code base (2024), which uses *Marabou* (2019) for NN-constrained SMT. For MaxSMT we use *Z3* (2008). All experiments were run on machines with AMD EPYC 9654 processors at 2.4GHz with time and memory limits of 12h and 4GB. Our tool and all experiments are available online.

Benchmarks. We use probabilistic versions of Vea's benchmarks (2022). These benchmarks are adaptations of the planning domains Blocksworld, Puzzle, and Transport encoded in the automata language JANI (Budde et al. 2017).

In Blocksworld and Puzzle, Vea already train their policies in a probabilistic environment. For verification, they abstract probabilistic transitions to non-deterministic ones, amounting to a worst-case analysis. We perform probabilistic verification. In Blocksworld, the policy is unsafe if the number of blocks on the table exceeds a fixed limit. Blocks probabilistically drop on the table. In Puzzle, unsafe states are specified in terms of unsafe tile positions. Tiles may probabilistically move to an unsafe position. Vea's Transport version is non-probabilistic and we add probabilistic behavior: Whenever the truck loads more than one package, it may drop probabilistically. Dropping a package is unsafe.

State space enumeration and thereby naive Q-learning, is infeasible on all these benchmarks. Neural policies are trained using deep Q-learning (Mnih et al. 2015). They are provided in PYTORCH (Ansel et al. 2024) format. Each policy has two hidden layers of size 16, 32 or 64 respectively. There are policies that are, vs. ones that are not, aware of move costs. Vea consider policies with vs. without *applicability filter*, i.e., pre-filtering the policy selection in each state to the applicable actions (2024).

Configurations. We run prob-CEGAR-PPA-inc for automated bound derivation and prob-CEGAR-PPA-p_u for verification of specific p_u. We set $p_\varepsilon = 10^{-5}$. The default configuration for linear search on p_u is $p_\Delta = 0.05$ and $p_u^{init} = 0$ (initial p_u). We also compare prob-PPA, computing the reachable fragment of probabilistic PPA for a fixed predicate set, with non-probabilistic PPA (non-prob-PPA) (Vea 2022). The latter abstracts probabilistic transitions to non-deterministic ones, introducing one (non-probabilistic) transition per (probabilistic) outcome.

Encoding into SOA model checkers. We compare with the probabilistic model checkers of the *Quantitative Verification Competition* (Budde et al. 2021), specifically STORM (2022), ePMC (2014), MODEST (2014), PRISM (2011) and PET (2022). These tools do not natively support neural policies π. Instead, the NN is encoded as an individual automaton and composed with the automata network that encodes the PTS. System transitions are controlled by π via synchronization. The NN output is computed via internal edges of the NN automaton. ePMC and MODEST support a straightforward encoding of the NN with neuron values encoded as real-valued variables. The other checkers do not allow real-valued variables. Here, we discretize the NN with finite precision. Neuron values are encoded as integer variables. The NN output is computed using rational arithmetic. This discretization does not guarantee to mimic the true π-restricted system faithfully, but is the best possible basis for a performance comparison given the checkers' limitations. We provide a detailed description in the TA.

We experiment with various configurations to compute the maximal unsafety probability $maxPr(\Theta^\pi, \phi_0, \phi_u)$. ePMC constructs the ($\pi$-restricted) state space *explicitly* and runs value iteration. PET performs *partial exploration* of Θ^π. We run MODEST's *explicit* engine and its FRET-LRTDP implementation. For STORM and PRISM, we experiment with their *explicit* and *symbolic* (binary decision diagram) engines, as well as explicit-symbolic *hybrids*. Additionally, we run STORM's *abstraction-refinement* based on a variant of (game-based) predicate abstraction similar to prob-CEGAR.

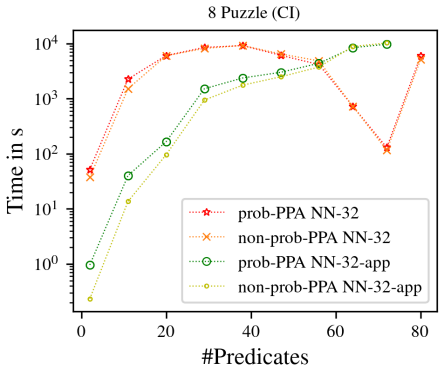

Figure 1: Abstract state space computation prob-PPA vs. non-prob-PPA. Predicate set scales as per Vea (2022).

Results: Computing $\Theta^\pi_\mathcal{P}$. Figure 1 compares the time to compute the reachable fragment of $\Theta^\pi_\mathcal{P}$ for prob-PPA vs. non-prob-PPA over predicate sets of increasing size. The

figure shows results for a selection of benchmarks; similar trends are observed in all benchmarks. prob-PPA loses little performance compared to non-prob-PPA. So there is only a small overhead incurred by enumerating abstract successor distributions $\mu_{\mathcal{P}}$ instead of abstract successor states $s'_{\mathcal{P}}$. This is presumably because the distribution size $|Supp(\mu_{\mathcal{P}})|$ is bounded by the update support $Supp(\bar{u})$ per operator o, which is typically small in practice.

Benchmark	NN	App	$[P_u^{lo}, P_u^{up}]$	p_u^{last}	Time	CE	$ups_{\mathcal{P}}^{max}$
4 Blocks (CI)	16	×	[0.37, 0.41]	$\ni 0.4$	18	1	75
	16	✓	[0.41, 0.65]	$\ni 0.45$	-	43036	74220
	32	×	[0.25, 0.27]	> 0.25	33	1	20
	32	✓	[0.19, 1]	$\ni 0.2$	-	42872	179662
	64	×	[0.17, 0.19]	≤ 0.2	7553	1	5
6 Blocks (CI)	16	×	[0.95, 1]	> 0.95	79	1	29
	16	✓	[0.95, 1]	> 0.95	321	1	29
	32	×	[0.27, 0.3]	≤ 0.3	14227	1	258
8 Blocks (CI)	16	×	[0.95, 1]	> 0.95	454	1	29
	16	✓	[0.95, 1]	> 0.95	3396	1168	20771
	32	×	[0.95, 1]	> 0.95	31376	1	29
8 Puzzle (CI)	16	×	[0.95, 1]	> 0.95	5434	3	10
	32	×	[0.1, 1]	$\ni 0.1$	-	43	2
	32	✓	[0.1, 1]	$\ni 0.1$	-	191	2
	64	✓	[0.95, 1]	> 0.95	9630	9603	209271
Transport	16	×	[0.1, 1]	$\ni 0.1$	-	113	2
	16	✓	[0.1, 1]	$\ni 0.1$	-	593	2
	32	×	[0.1, 1]	$\ni 0.1$	-	1688	2
	32	✓	[0.1, 1]	$\ni 0.1$	-	9193	2
4 Blocks (CA)	16	×	[0.34, 0.36]	$\ni 0.35$	44	1	305
	16	✓	[0.34, 0.42]	$\ni 0.35$	-	43054	104411
	32	×	[0.27, 0.28]	≤ 0.3	1145	393	6853
	32	✓	[0.17, 0.44]	$\ni 0.2$	-	41818	162006
6 Blocks (CA)	16	×	[0.86, 1]	$\ni 0.9$	-	43043	420997
	32	×	[0.95, 1]	> 0.95	4121	1	29
	32	✓	[0.95, 1]	> 0.95	38289	1	29
8 Blocks (CA)	16	×	[0.99, 1]	> 0.95	5692	1	3
	32	×	[0.84, 1]	$\ni 0.85$	-	15661	224957
8 Puzzle (CA)	16	×	[0.1, 1]	$\ni 0.1$	-	10	2
	16	✓	[0.1, 1]	$\ni 0.1$	-	28	2
	32	×	[0.1, 1]	$\ni 0.1$	-	33	2
	32	✓	[0.1, 1]	$\ni 0.1$	-	56	2

Table 1: Results for prob-CEGAR-PPA-inc over different benchmarks distinguishing cost-aware (CA) and cost-ignoring (CI) policies, and policies with/out applicability-filter. $[P_u^{lo}, P_u^{up}]$: derived probability bounds. p_u^{last}: unsafety bound p_u at termination. $[P_u^{lo}, P_u^{up}]$ verifies ($\leq$), falsifies ($>$), or contains ($\ni$) p_u^{last}. Total runtime in seconds. - indicates runs that exceed the resource limit (12h, 4 GB). CE: time spent on counterexample analysis. $ups_{\mathcal{P}}^{max}$: size of the maximal abstract unsafe path set constructed.

Results: Automated bound derivation. Table 1 shows results for prob-CEGAR-PPA-inc. It terminates with a complete analysis on 16 out of the 32 benchmark instances. On the remaining instances, it derives tightened probability bounds – in 5 cases with an interval size smaller than 0.3 – demonstrating its anytime behavior.

Counterexample analysis (CeAna) is often a major bottleneck. On 6 instances (5 timeouts) more than 50% of the time is spent on CeAna, showcasing the complexity of probabilistic counterexample analysis. One source of complexity during CeAna are costly π-spuriousness checks. On Transport instances, e.g., almost the entire CE-time is spent on π-path checks. On many instances, however, the complexity arises from the size of the abstract unsafe path set $ups_{\mathcal{P}}^{max}$. On 8 instances (6 timeouts), $ups_{\mathcal{P}}^{max}$ exceeds 10^4. Notably, on all instances $ups_{\mathcal{P}}^{max}$ is constructed while abstract probability is insufficient $Pr(ups_{\mathcal{P}}) \leq p_u$, and hence independent of minimal path probability p_ε.

CeAna is particularly challenging for policies with applicability filtering (9 timeouts out of 13). This aligns with findings for non-probabilistic PPA (2024): while applicability filtering simplifies learning, it adds complexity to abstract state space computation. In probabilistic PPA, it also adds complexity to CeAna: abstract counterexamples here are prone to induce many low-probability paths (6 out of 8 instances with large $ups_{\mathcal{P}}^{max}$).

Table 1 shows results for $(p_\Delta, p_u^{init}) = (0.05, 0)$. We also experimented with $p_\Delta \in \{0.01, 0.1\}$ and $(p_\Delta, p_u^{init}) = (0.05, 1)$. Overall, the runtime performance of all configurations is similar. We show detailed results in the TA.

p_u	4 Blocks CI 64			4 Blocks CA 32			4 Blocks CA 32 (app)		
	safe	PPA-p_u	PPA-inc	safe	PPA-p_u	PPA-inc	safe	PPA-p_u	PPA-inc
0.0	×	**7119**	7553	×	**345**	375	×	**901**	920
0.05	×	**7328**	7553	×	**592**	631	×	**903**	920
0.1	×	**7149**	7553	×	**399**	631	×	**906**	920
0.2	✓	**7130**	7553	×	**428**	667	?	-	-
0.3	✓	**7258**	7553	✓	**805**	1145	?	-	-
0.5	✓	**7279**	7553	✓	**399**	1145	✓	-	**2056**

Table 2: Verification time of specific p_u with (prob-CEGAR)-PPA-p_u vs. *during* (prob-CEGAR)-PPA-inc.

Results: Verification of p_u. Table 2 compares the verification of a specific bound p_u with (prob-CEGAR)-PPA-p_u vs. the verification time of that p_u *during* (prob-CEGAR)-PPA-inc. We show results for a selection of benchmarks; similar trends are observed on all benchmarks.

Verification tailored to a specific p_u can be significantly faster, in particular on instances where p_u is significantly larger than the concrete unsafety probability (4 Blocks, CA, 32, 0.5). That said, verification time during PPA-inc is often on par. This demonstrates that linear search on p_u, if conducted incrementally, is practical. On instances with p_u close to the concrete unsafety probability, PPA-inc can even make verification feasible in the first place (4 Blocks, CA, 32, app, $p_u = 0.5$).

Results: Comparison with SOA model checkers. *The probabilistic model checkers are almost universally unsuccessful.* All configurations exceed either time or memory limits on all benchmark instances. The only exception is STORM's explicit engine which successfully terminates on 6 Blocks (CI) NN 16 without app-filter after 180 seconds. This is more than double the runtime of prob-CEGAR-PPA-inc with $p_\Delta = 0.01, 0.05$ or 0.10 (less than 80 seconds).

Conclusion

We have extended PPA to probabilistic systems. Our approach enables both, verification of upper bounds on unsafety probability, and automated derivation of interval bounds. Our experiments show that probabilistic PPA is superior to encodings into non-NN tailored model checkers.

Future work may further strengthen our method. Compact abstract path set representations (Han et al. 2009) might improve counterexample analysis where enumeration is infeasible. Another important future direction for PPA is liveness verification, e.g., that π will eventually achieve a goal.

Acknowledgments

This work was funded by DFG Grant 389792660 as part of TRR 248 – CPEC (https://perspicuous-computing.science).

References

Amir, G.; Corsi, D.; Yerushalmi, R.; Marzari, L.; Harel, D.; Farinelli, A.; and Katz, G. 2023. Verifying Learning-Based Robotic Navigation Systems. In *TACAS*, volume 13993 of *LNCS*, 607–627. Springer.

Ansel, J.; Yang, E. Z.; He, H.; Gimelshein, N.; Jain, A.; Voznesensky, M.; Bao, B.; Bell, P.; Berard, D.; Burovski, E.; Chauhan, G.; Chourdia, A.; Constable, W.; Desmaison, A.; DeVito, Z.; Ellison, E.; Feng, W.; Gong, J.; Gschwind, M.; Hirsh, B.; Huang, S.; Kalambarkar, K.; Kirsch, L.; Lazos, M.; Lezcano, M.; Liang, Y.; Liang, J.; Lu, Y.; Luk, C. K.; Maher, B.; Pan, Y.; Puhrsch, C.; Reso, M.; Saroufim, M.; Siraichi, M. Y.; Suk, H.; Zhang, S.; Suo, M.; Tillet, P.; Zhao, X.; Wang, E.; Zhou, K.; Zou, R.; Wang, X.; Mathews, A.; Wen, W.; Chanan, G.; Wu, P.; and Chintala, S. 2024. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 ASPLOS 2024, La Jolla, CA, USA, 27 April 2024- 1 May 2024*, 929–947. ACM.

Bacci, E.; and Parker, D. 2020. Probabilistic Guarantees for Safe Deep Reinforcement Learning. In *Formal Modeling and Analysis of Timed Systems - 18th International Conference, FORMATS 2020, Vienna, Austria, September 1-3, 2020 Proceedings*, volume 12288 of *LNCS*, 231–248. Springer.

Bertsekas, D. P. 1995. *Dynamic Programming and Optimal Control*. Athena Scientific.

Bjørner, N. S.; and Phan, A. 2014. νZ - Maximal Satisfaction with Z3. In *6th International Symposium on Symbolic Computation in Software Science, SCSS 2014, Gammarth, La Marsa, Tunisia, December 7-8, 2014*, volume 30 of *EPiC Series in Computing*, 1–9. EasyChair.

Budde, C. E.; Dehnert, C.; Hahn, E. M.; Hartmanns, A.; Junges, S.; and Turrini, A. 2017. JANI: Quantitative Model and Tool Interaction. In *TACAS*, volume 10206 of *LNCS*, 151–168.

Budde, C. E.; Hartmanns, A.; Klauck, M.; Kretínský, J.; Parker, D.; Quatmann, T.; Turrini, A.; and Zhang, Z. 2021. On Correctness, Precision, and Performance in Quantitative Verification - QComp 2020 Competition Report. In *Intl. Symp. on Leveraging Applications of Formal Methods, ISoLA*, volume 12479 of *LNCS*, 216–241. 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.

de Moura, L.; and Bjørner, N. 2008. Z3: An Efficient SMT Solver. In *TACAS*, volume 4963 of *LNCS*, 337–340. Springer.

Graf, S.; and Saïdi, H. 1997. Construction of Abstract State Graphs with PVS. In *CAV*, volume 1254 of *LNCS*, 72–83. Springer.

Hahn, E. M.; Li, Y.; Schewe, S.; Turrini, A.; and Zhang, L. 2014. iscasMc: A Web-Based Probabilistic Model Checker. In *Formal Methods, World Congress, FM*, volume 8442 of *Lecture Notes in Computer Science*, 312–317. Springer.

Han, T.; and Katoen, J. 2007. Counterexamples in Probabilistic Model Checking. In *TACAS*, volume 4424 of *LNCS*, 72–86. Springer.

Han, T.; Katoen, J.; and Damman, B. 2009. Counterexample Generation in Probabilistic Model Checking. *IEEE Trans. Software Eng.*, 35(2): 241–257.

Hartmanns, A.; and Hermanns, H. 2014. The Modest Toolset: An Integrated Environment for Quantitative Modelling and Verification. In *TACAS*, volume 8413 of *LNCS*, 593–598. Springer.

Hensel, C.; Junges, S.; Katoen, J.; Quatmann, T.; and Volk, M. 2022. The probabilistic model checker Storm. *Intl. J. on Soft. Tools for Tech. Transfer*, 24(4): 589–610.

Hermanns, H.; Wachter, B.; and Zhang, L. 2008. Probabilistic CEGAR. In *CAV*, volume 5123 of *LNCS*, 162–175. Springer.

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.; 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 *CAV*, volume 11561 of *LNCS*, 443–452. Springer.

Katz, S. M.; Julian, K. D.; Strong, C. A.; and Kochenderfer, M. J. 2023. Generating probabilistic safety guarantees for neural network controllers. *Mach. Learn.*, 112(8): 2903–2931.

Kwiatkowska, M. Z.; Norman, G.; and Parker, D. 2011. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *CAV*, volume 6806 of *LNCS*, 585–591. Springer.

Lopez, D. M.; Choi, S. W.; Tran, H.; and Johnson, T. T. 2023. NNV 2.0: The Neural Network Verification Tool. In *CAV*, volume 13965 of *LNCS*, 397–412. Springer.

Meggendorfer, T. 2022. PET - A Partial Exploration Tool for Probabilistic Verification. In *Automated Technology for*

Verification and Analysis, Intl. Symp., ATVA, volume 13505 of *LNCS*, 320–326. Springer.

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.

Stahlberg, S.; Bonet, B.; and Geffner, H. 2022. Learning General Optimal Policies with Graph Neural Networks: Expressive Power, Transparency, and Limits. In *ICAPS*, 629–637. AAAI Press.

Steinmetz, M.; Hoffmann, J.; and Buffet, O. 2016. Goal Probability Analysis in Probabilistic Planning: Exploring and Enhancing the State of the Art. *JAIR*, 57: 229–271.

Strong, C. A.; Wu, H.; Zeljic, A.; Julian, K. D.; Katz, G.; Barrett, C. W.; and Kochenderfer, M. J. 2023. Global optimization of objective functions represented by ReLU networks. *Mach. Learn.*, 112(1): 3685–3712.

Vinzent, M.; and Hoffmann, J. 2024. Neural Action Policy Safety Verification: Applicablity Filtering. In *ICAPS*, 607–612. AAAI Press.

Vinzent, M.; Sharma, S.; and Hoffmann, J. 2023. Neural Policy Safety Verification via Predicate Abstraction: CEGAR. In *AAAI*, 15188–15196. AAAI Press.

Vinzent, M.; Steinmetz, M.; and Hoffmann, J. 2022. Neural Network Action Policy Verification via Predicate Abstraction. In *ICAPS*. AAAI Press.

Wachter, B.; Zhang, L.; and Hermanns, H. 2007. Probabilistic Model Checking Modulo Theories. In *Fourth International Conference on the Quantitative Evaluaiton of Systems (QEST 2007), 17-19 September 2007, Edinburgh, Scotland UK*, 129–140. IEEE Computer Society.

Wang, Y.; Zhou, W.; Fan, J.; Wang, Z.; Li, J.; Chen, X.; Huang, C.; Li, W.; and Zhu, Q. 2024. POLAR-Express: Efficient and Precise Formal Reachability Analysis of Neural-Network Controlled Systems. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 43(3): 994–1007.