Probabilistic Safety Verification of Neural Policies via Predicate Abstraction Technical Appendix

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

Additional Experimental Results

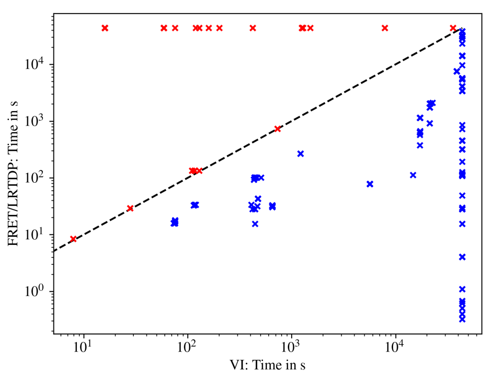


Figure 2: Abstract MaxProb with FRET-LRTDP vs. VI (value iteration). For each benchmark instance there is a data point for the verification time of each bound p_u in $\{0, 0.05, \dots 0.95\}$ during prob-CEGAR-PPA-inc with $(p_\Delta, p_u^{init}) = (0.05, 0)$, indicating whether FRET-LRTDP or VI is faster.

Abstract MaxProb. Figure 2 compares abstract MaxProb search with FRET-LRTDP vs. VI (value iteration), specifically the verification time of individual p_u during prob-CEGAR-PPA-inc. FRET-LRTDP largely outperforms VI. Constructing the reachable fragment of $\Theta^\pi_\mathcal{P}$ is often too expensive. FRET-LRTDP mitigates this thanks to incremental exploration of $\Theta^\pi_\mathcal{P}$. VI dominates on some instances. We observe that, thanks to its complete exploration of $\Theta^\pi_\mathcal{P}$, VI may select abstract counterexamples with short and realizable paths in early CEGAR iterations.

Automated bound derivation. Table 3 shows results for prob-CEGAR-PPA-inc for different configurations of (p_Δ, p_u^{init}). Overall, the runtime performance of all configurations is similar, especially for $p_u^{init} = 0$. This indicates (1) there is a sweet spot during CEGAR after which $[P_u^{lo}, P_u^{up}]$ can be tightened heavily (2) since linear search is conducted incrementally the impact of p_Δ is alleviated. That said, $p_\Delta = 0.01$ never drastically faster than all other configuration, but can be significantly slower (e.g., 4 Blocks, CA, 32) showing that overly conservative p_Δ may still be counter-beneficial. $(p_\Delta = 0.05, p_u^{init} = 1)$ has a smaller coverage (12 instances) than $(p_\Delta = 0.05, p_u^{init} = 0)$ (16 instances). For large (initial) p_u, CeAna is more prone to enumerate an infeasible number of paths until the accumulated abstract probability $Pr(ups_\mathcal{P})$ exceeds p_u. In contrast, for small p_u, CeAna may proceed to the computation of $maxPr[ups_\mathcal{P}]$; establishing non-trivial P_u^{lo}. However, there are also instances where $p_u^{init} = 1$ dominates $p_u^{init} = 0$.

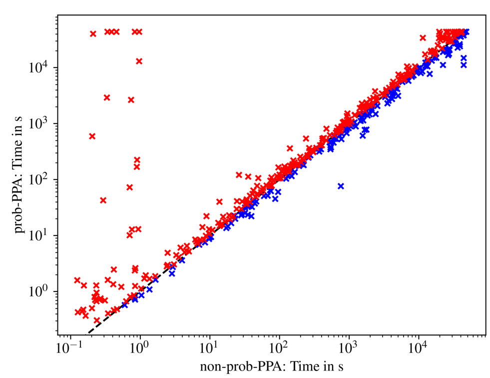


Figure 3: Abstract state space computation prob-PPA vs. non-prob-PPA, indicating whether prob-PPA or non-prob-PPA is faster. Predicate set scales as per Vea (2022).

Computing $\Theta^\pi_\mathcal{P}$. Figure 3 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. It completes the results presented in the main text (Figure 1), showing results for all benchmarks. In line with our observations in the main text, prob-PPA overall loses little performance in abstract state space building compared to non-prob-PPA. There are outliers for very coarse abstractions, i.e., small predicate sets $\mathcal{P}$, where non-prob-PPA finishes in seconds while prob-PPA takes hours. For such $\mathcal{P}$, SMT under π is notoriously expensive (Vea 2022), and, thereby, more prone to outliers: Some SMT calls take drastically longer than others. The additional intricacy in prob-PPA leads to an increase of such outliers.

Benchmark	NN	App	$p_\Delta = 0.01$		$p_\Delta = 0.05$		$p_\Delta = 0.05, p_u^{init} = 1$		$p_\Delta = 0.1$	
			$[P_u^{lo}, P_u^{up}]$	Time	$[P_u^{lo}, P_u^{up}]$	Time	$[P_u^{lo}, P_u^{up}]$	Time	$[P_u^{lo}, P_u^{up}]$	Time
4 Blocks (CI)	16	×	[0.4, 0.41]	**15**	[0.37, 0.41]	18	[0.41, 1]	-	[0.37, 0.41]	16
	16	✓	[0.41, 0.64]	-	[0.41, 0.65]	-	[0.41, 0.66]	-	[0.41, 0.65]	-
	32	×	[0.27, 0.27]	**29**	[0.25, 0.27]	33	[0.25, 0.27]	30	[0.2, 0.27]	**29**
	32	✓	[0.19, 1]	-	[0.19, 1]	-	[0.19, 1]	-	[0.19, 1]	-
	64	×	[0.19, 0.19]	7200	[0.17, 0.19]	7553	[0.19, 0.19]	**7157**	[0.17, 0.26]	7560
6 Blocks (CI)	16	×	[0.99, 1]	**66**	[0.95, 1]	79	[0.95, 1]	67	[0.9, 1]	67
	16	✓	[0.99, 1]	313	[0.95, 1]	321	[0.95, 1]	**310**	[0.9, 1]	315
	32	×	[0.27, 0.27]	12397	[0.27, 0.3]	14227	[0.27, 0.9]	-	[0.27, 0.3]	**11762**
8 Blocks (CI)	16	×	[0.99, 1]	392	[0.95, 1]	454	[0.95, 1]	**383**	[0.9, 1]	386
	16	✓	[0.99, 1]	3493	[0.95, 1]	**3396**	[0, 1]	-	[0.9, 1]	3457
	32	×	[0.99, 1]	27260	[0.95, 1]	31376	[0.95, 1]	**25743**	[0.9, 1]	25873
8 Puzzle (CI)	16	×	[0.99, 1]	4938	[0.95, 1]	5434	[0.95, 1]	33433	[0.92, 1]	**4908**
	32	×	[0.1, 1]	-	[0.1, 1]	-	[0.1, 1]	-	[0.1, 1]	-
	32	✓	[0.1, 1]	-	[0.1, 1]	-	[0.1, 1]	-	[0.1, 1]	-
	64	✓	[0.95, 1]	-	[0.95, 1]	9630	[0.95, 1]	13135	[0.9, 1]	**48**
Transport	16	×	[0.1, 1]	-	[0.1, 1]	-	[0, 1]	-	[0.1, 1]	-
	16	✓	[0.1, 1]	-	[0.1, 1]	-	[0, 1]	-	[0.1, 1]	-
	32	×	[0.1, 1]	-	[0.1, 1]	-	[0, 1]	-	[0.1, 1]	-
	32	✓	[0.1, 1]	-	[0.1, 1]	-	[0, 1]	-	[0.1, 1]	-
4 Blocks (CA)	16	×	[0.34, 0.34]	44	[0.34, 0.36]	44	[0.02, 1]	-	[0.26, 0.36]	**41**
	16	✓	[0.34, 0.42]	-	[0.34, 0.42]	-	[0.03, 1]	-	[0.34, 0.47]	-
	32	×	[0.27, 0.27]	19690	[0.27, 0.28]	1145	[0.27, 0.3]	**422**	[0.27, 0.3]	785
	32	✓	[0.22, 0.44]	-	[0.17, 0.44]	-	[0.14, 1]	-	[0.14, 1]	-
6 Blocks (CA)	16	×	[0.86, 1]	-	[0.86, 1]	-	[0, 1]	-	[0.86, 1]	-
	32	×	[0.99, 1]	**3589**	[0.95, 1]	4121	[0.95, 1]	3591	[0.9, 1]	3637
	32	✓	[0.99, 1]	**38160**	[0.95, 1]	38289	[0.95, 1]	40772	[0.9, 1]	39409
8 Blocks (CA)	16	×	[0.99, 1]	5283	[0.99, 1]	5692	[0.99, 1]	5029	[0.91, 1]	**4866**
	32	×	[0.84, 1]	-	[0.84, 1]	-	[0, 1]	-	[0.8, 1]	-
8 Puzzle (CA)	16	×	[0.1, 1]	-	[0.1, 1]	-	[0.1, 1]	-	[0.1, 1]	-
	16	✓	[0.1, 1]	-	[0.1, 1]	-	[0.1, 1]	-	[0.1, 1]	-
	32	×	[0.1, 1]	-	[0.1, 1]	-	[0.1, 1]	-	[0.1, 1]	-
	32	✓	[0.1, 1]	-	[0.1, 1]	-	[0.1, 1]	-	[0.1, 1]	-

Table 3: Results for prob-CEGAR-PPA-inc with FRET-LRTDP for different configurations of (p_Δ, p_u^{init}). Default $p_u^{init} = 0$.

Verification of p_u. Figure 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. It completes the results presented in the main text (Table 2), showing results for all benchmarks. The results are in line with the observations in the main text. PPA-p_u tends to dominate and can be significantly faster. That said, PPA-inc is overall competitive and can, in some cases, even make verification feasible in the first place (outliers).

Proofs – Probabilistic PPA

Wachter et al. (2007) show that probabilistic predicate abstraction preserves satisfaction of safe PCTL formulae (Bianco and de Alfaro 1995).

Theorem (Wachter et al. 2007)**.** If $\Theta_{\mathcal{P}}$ satisfies a safe PCTL formula Φ, then Θ also satisfies Φ.

Proposition 2 (safety in $\Theta_{\mathcal{P}}$ implies safety in Θ) follows as a special case of Theorem . Proposition 4 (safety in $\Theta^\pi_{\mathcal{P}}$ implies safety in Θ^π) follows analogously.

Probabilistic Abstract State Expansion

Algorithm 3 shows pseudocode for probabilistic abstract state expansion. The algorithm is an adaption of state expansion for non-probabilistic PPA (Vea 2022). Specifically, we deploy applicability (line 2) and $\mathcal{T}$-transition tests (line 30) to avoid costly SMT tests under π. Additionally, we lift the latter to the probabilistic case (line 15). `EnumPaDist` (line 7) shows the enumeration of abstract successor distribution (candidates). We first compute abstract successor states for each individual update $u \in Supp(\bar{u})$ (line 9) using Vea's enumeration procedure (line 22). For the abstract transition problem under π (line 5), we apply Vea's enhancements to efficiently handle π: continuous relaxation of integer state variable to use NN-tailored SMT solvers (Katz et al. 2019) embedded into a branch & bound framework. If the transition exists, it is "processed" for MaxProb search (line 6).

Proofs – prob-CEGAR-PPA

prob-CEGAR (Hermanns et al. 2008) maintains a set of abstract unsafe paths $ups_{\mathcal{P}} \subseteq Path(A_{\mathcal{P}}, s^0_{\mathcal{P}}, \phi_u)$. Han et al. (2009) show that for all probabilistic counterexamples (A, s^0) – concrete or abstract – there exist a finite subset of unsafe paths (with decreasing probability) that "witnesses" the violation of p_u. We use a slightly more general formulation for any (infinite) subset induced by (A, s^0). The proof is adapted in a straight-forward manner.

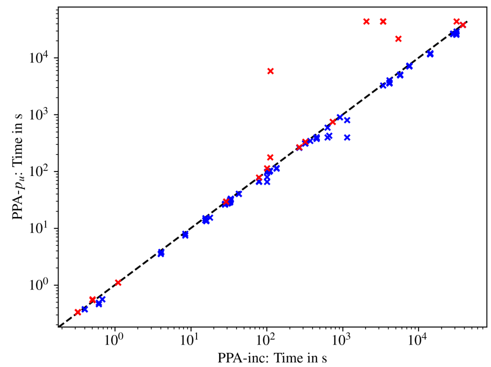


Figure 4: Verification time of $p_u \in 0, 0.05, 0.1, 0.2, 0.3, 0.5$ with (prob-CEGAR)-PPA-p_u vs. *during* (prob-CEGAR)-PPA-inc, indicating whether PPA-p_u or PPA-inc is faster.

Lemma 9. *Let $ups \subseteq Path(A, s^0, \phi_u)$. $Pr(ups) > p_u$ iff there exists a finite set of unsafe paths $ups' \subseteq Path(A, s^0, \phi_u)$ such that $Pr(ups') > p_u$ and for all $\sigma' \in ups', \sigma \in ups \setminus ups'$ it holds $Pr(\sigma') \geq Pr(\sigma)$.*

The original result then follows as a corollary, i.e., $Pr(A, s^0, \phi_u) > p_u$ iff there exists finite $ups \subseteq Path(A, s^0, \phi_u)$ (with decreasing probability) such that $Pr(ups) > p_u$.

Termination of CeAna. We first prove some lemma.

Lemma 10. *Whenever Algorithm 1 enters loop (line 11 - 21) it holds $P_{max} \leq p_u$ and can-add$(ups_{\mathcal{P}})$.*

Proof. The statement holds initially. Moreover, whenever $\mathcal{T}$-analysis iterates it holds $P_{max} \leq p_u$ (line 19) and *can-add*$(ups_{\mathcal{P}})$ (line 20). □

Lemma 11. *For any $p_\varepsilon > 0$. The set $ups_{\mathcal{P}} := \{\sigma_{\mathcal{P}} \in Path(A_{\mathcal{P}}, s^0_{\mathcal{P}}, \phi_u) \mid Pr(\sigma_{\mathcal{P}}) \geq p_\varepsilon\}$ is finite.*

Proof. Proof by contradiction. For $ups_{\mathcal{P}}$ with infinitely many paths of probability at least p_ε, $Pr(ups_{\mathcal{P}})$ diverges. However, $Pr(ups) \leq Pr(A_{\mathcal{P}}, s^0_{\mathcal{P}}, \phi_u) \leq 1$. Contradiction. □

Proof of Theorem 5 . It suffices to show that $\mathcal{T}$-analysis (line 11 - 21) terminates. By Lemma 10, in each iteration loop (line 12 - 17) is entered. By Lemma 9 and Lemma 11, $Pr(ups_{\mathcal{P}}) > p_u$ and $\min_{\sigma_{\mathcal{P}} \in ups_{\mathcal{P}}} Pr(\sigma_{\mathcal{P}}) < p_\varepsilon$ for finite $ups_{\mathcal{P}}$ respectively. Hence loop (line 12 - 17) has only finitely many iterations over an invocation of `CeAna`. Hence, $\mathcal{T}$-analysis terminates. □

Correctness of CeAna.

Lemma 12. *During an invocation of* `CeAna` *it holds $maxPr[ups_{\mathcal{P}}](\Theta) \leq P_{max} \leq Pr(ups)$.*

Proof. The statement holds at initialization and is preserved as an invariant: (line 16, 17): Let $ups'_{\mathcal{P}} = ups_{\mathcal{P}} \cup \{\sigma_{\mathcal{P}}\}$. Let $P'_{max} = P_{max} + Pr(\sigma_{\mathcal{P}})$. $maxPr[ups'_{\mathcal{P}}] \leq maxPr[ups_{\mathcal{P}}] + Pr(\sigma_{\mathcal{P}}) \leq P_{max} + Pr(\sigma_{\mathcal{P}}) = P'_{max}$. $P'_{max} = P_{max} + Pr(\sigma_{\mathcal{P}}) \leq Pr(ups_{\mathcal{P}}) + Pr(\sigma_{\mathcal{P}}) = Pr(ups'_{\mathcal{P}})$.
(line 18): Trivial. □

Lemma 12 shows that P_{max} is an upper bound on the maximal realizable probability as intended.

Lemma 13. *While* `CeAna` *iterates loop (line 12 - 17) it holds $P_{max} + P_{unused} > p_u$.*

Proof. The statement holds initially and is required whenever $\mathcal{T}$-analysis iterates (line 20). Moreover, while loop (line 12 - 17) continues the sum $P_{max} + P_{unused}$, and thereby the statement, are preserved (line 16, 17). □

Lemma 13 guarantees that CeAna behaves well in that whenever path computation is invoked (line 13) the $|ups_{\mathcal{P}}|+1$-th path actually exists.

Proof of Theorem 6 . By π-all realization of $ups^*_{\mathcal{P}}$ (line 25), there exist $s^0 \in [s^0_{\mathcal{P}}] \cap [\phi_0]$ such that for each $\sigma_{\mathcal{P}} \in ups^*_{\mathcal{P}}$, there exists $\sigma \in [\sigma_{\mathcal{P}} \wedge \phi_u](\Theta^\pi)$ with $s^0(\sigma) = s^0$, and thereby $\sigma \in Path([A^\pi_{\mathcal{P}}], s^0, \phi_u)$. Let $ups \subseteq Path([A^\pi_{\mathcal{P}}], s^0, \phi_u)$ denote the corresponding concrete path set. By construction $Pr(ups) = Pr(ups^*_{\mathcal{P}}) > p_u$ and hence π is unsafe. □

Termination of prob-CEGAR-PPA.

Lemma 14. *Let $\mathcal{S}^0 \subseteq \mathcal{S}$ such that $\left|\mathcal{S}^0\right| \geq 2$. Let*

$$\bigcup_{v \in \mathcal{V}} \bigcup_{i \in 1\ldots,|\mathcal{S}^0(v)|-1} \{v \leq \mathcal{S}^0_i(v)\} \subseteq \mathcal{P},$$

where $\mathcal{S}^0(v) = \{s(v) \mid s \in \mathcal{S}^0\}$ and $\mathcal{S}^0_i(v) \in \mathcal{S}^0(v)$ denotes the i-th smallest value. For all distinct $s, t \in \mathcal{S}^0$ it holds $s|_{\mathcal{P}} \neq t|_{\mathcal{P}}$.

Proof. $s \neq t$ and thereby $s(v) \neq t(v)$ for some $v \in \mathcal{V}$. Let, w.l.o.g., $s(v) < t(v)$. There exists $(v \leq c) \in \mathcal{P}$ such that $s \models v \leq c$ while $\neg(t \models v \leq c)$. Hence, $s|_{\mathcal{P}} \neq t|_{\mathcal{P}}$. □

Lemma 15. *Whenever* `CeAna` *invokes refinement for (ii-a) (line 21) (or for (ii-b) (line 26)) there exist distinct $\sigma_{\mathcal{P}}, \sigma'_{\mathcal{P}} \in ups_{\mathcal{P}}$ such that there exist unsafe concretizations $\sigma \in [\sigma_{\mathcal{P}} \wedge \phi_u](\Theta)$ and $\sigma' \in [\sigma'_{\mathcal{P}} \wedge \phi_u](\Theta)$ (or $\sigma \in [\sigma_{\mathcal{P}} \wedge \phi_u](\Theta^\pi)$ and $\sigma' \in [\sigma'_{\mathcal{P}} \wedge \phi_u](\Theta^\pi)$) with $s^0(\sigma) \neq s^0(\sigma')$.*

Proof. (ii-a): Consider maximal $\mathcal{T}$-realizable subset $ups^*_{\mathcal{P}} \subseteq ups_{\mathcal{P}}$ with maximizing start state s^0_*. By inner loop condition (line 12), Lemma 12, and definition of *can-add*$(ups_{\mathcal{P}})$, refinement for (ii-a) is invoked only once $Pr(ups_{\mathcal{P}}) > p_u$ while $maxPr[ups_{\mathcal{P}}](\Theta) \leq p_u$ (line 19). Hence, $ups_{\mathcal{P}} \setminus ups^*_{\mathcal{P}}$ is non-empty. Let $\sigma_{\mathcal{P}} \in ups^*_{\mathcal{P}}$ and $\sigma'_{\mathcal{P}} \in ups_{\mathcal{P}} \setminus ups^*_{\mathcal{P}}$. We know $\sigma_{\mathcal{P}}$ and $\sigma'_{\mathcal{P}}$ are $\mathcal{T}$-path realizable. Hence the statement holds for s^0_* and start state $s^0(\sigma')$ of any unsafe concretization $\sigma' \in [\sigma'_{\mathcal{P}} \wedge \phi_u]$.

(ii-b): The result for follows analogously. Since π-all realization fails (line 25), there exist $\sigma_{\mathcal{P}}, \sigma'_{\mathcal{P}} \in ups^*_{\mathcal{P}}$, which are π-path realizable (line 23), such that $s^0(\sigma) \neq s^0(\sigma')$ for any two unsafe concretization paths $\sigma \in [\sigma_{\mathcal{P}} \wedge \phi_u](\Theta^\pi)$ and $\sigma' \in [\sigma'_{\mathcal{P}} \wedge \phi_u](\Theta^\pi)$. □

Lemma 16. *In each iteration, CEGAR either terminates or strictly refines the abstraction $\mathcal{P} \subsetneq \mathcal{P}'$ in that there exist $s, t \in \mathcal{S}$ such that $s|_{\mathcal{P}} = t|_{\mathcal{P}}$ in the original abstraction while $s|_{\mathcal{P}'} \neq t|_{\mathcal{P}'}$ in the refined abstraction.*

Proof. The statement holds for refinement of spuriousness (i-a) and (i-b) as per Vea (2023) and by Lemma 14 and Lemma 15 for refinement of spuriousness (ii-a) and (ii-b). □

Proof of Theorem 7 . Let $s, t \in \mathcal{S}$. Since $\mathcal{S}$ is finite (bounded-integer state variables) and by Lemma 16, $s|_{\mathcal{P}} = t|_{\mathcal{P}}$ iff $s = t$ within finitely many iterations ($\Theta^\pi_{\mathcal{P}} \equiv \Theta^\pi$). Then also each abstract path is trivially realizable. Hence, either CEGAR returns SAFE or CeAna returns *REAL*, and hence CEGAR returns UNSAFE. □

CEGAR for Automated Bound Derivation

Algorithm 2: prob-CEGAR-PPA-inc.

Input: $\langle \mathcal{V}, \mathcal{L}, \mathcal{O} \rangle$, (ϕ_0, ϕ_u, p_u), $p_\Delta \in (0, 1]$.

1 $[P^{lo}_u, P^{up}_u] \leftarrow 0, 1$ // $maxPr(\Theta^\pi, \phi_0, \phi_u)$ interval.
2 $\mathcal{P} \leftarrow \{\phi_u\}$
3 **while** 1 **do**
4 $\quad (A^\pi_{\mathcal{P}}, s^0_{\mathcal{P}}) \leftarrow$ max-prob$(\langle \mathcal{V}, \mathcal{L}, \mathcal{O} \rangle, (\phi_0, \phi_u, p_u), \mathcal{P}, \pi)$
5 $\quad P^{up}_u \leftarrow \min(P^{up}_u, Pr(A^\pi_{\mathcal{P}}, s^0_{\mathcal{P}}, \phi_u))$
6 $\quad$ **if** $P^{up}_u - P^{lo}_u \leq p_\Delta$ **then return**
7 $\quad$ **while** $P^{up}_u \leq p_u$ **do** $p_u \leftarrow p_u - p_\Delta$ // Safe p_u.
8
9 $\quad ups_{\mathcal{P}}, P_{max} \leftarrow \emptyset, 0$
10 $\quad$ **while** 1 **do**
11 $\qquad result \leftarrow$ CeAna$(A^\pi_{\mathcal{P}}, s^0_{\mathcal{P}}, ups_{\mathcal{P}}, P_{max})$
12 $\qquad$ **if** *result* = *SPURIOUS* **then break**
13 $\qquad P^{lo}_u \leftarrow \max(P^{lo}_u, P_{max})$
14 $\qquad$ **if** $P^{up}_u - P^{lo}_u \leq p_\Delta$ **then return** // Unsafe p_u.
15 $\qquad$ **while** $P^{lo}_u > p_u$ **do** $p_u \leftarrow p_u + p_\Delta$

Algorithm 2 shows the adapted CEGAR loop with linear search on p_u for automated bound derivation (prob-CEGAR-PPA-inc). CEGAR continues until the interval $[P^{lo}_u, P^{up}_u]$ is sufficiently tightened (line 6, 14). P^{up}_u is updated to the (maximal) abstract unsafety probability (line 5). While p_u is safe, it is decreased (line 7). $ups_{\mathcal{P}}$ and P_{max} (line 9) are maintained over multiple invocations of CeAna within a CEGAR iteration. The linear search pauses and CEGAR iterates whenever CeAna returns *SPURIOUS* (line 12). If not, P^{lo}_u is updated according to the maximal π-realizable probability (line 13) computed by CeAna. p_u is increased (line 15).

By termination of prob-CEGAR-PPA-p_u, each p_u is proven safe or unsafe within finitely many iterations. When this case occurs, p_u is decreased or increased until $p_u < P^{up}_u$ or $P^{lo}_u \leq p_u$ respectively. Moreover, due to the termination condition (line 6, 14), it also holds $P^{lo}_u < p_u$ and $p_u < P^{up}_u$ respectively. In other words, p_u is updated until $p_u \in [P^{lo}_u, P^{up}_u)$ and hence safety is no longer deducible. Since P^{lo}_u and P^{up}_u are monotonically increasing and decreasing respectively, each p_u is iterated at most once during linear search. Hence, prob-CEGAR-PPA-inc terminates.

Neural Action Policy

A (ReLU) feed-forward *neural network* for Θ is a (real-valued) function

$$f_\pi \colon \mathcal{S} \to \mathbb{R}^{d_d}, s \mapsto f_d(\ldots f_2(f_1(s))),$$

where d denotes the number of layers in the NN, d_i for $i \in \{1, \ldots, d\}$ denotes the size of layer i, and

- $f_1 \colon \mathcal{S} \to \mathbb{R}^{d_1}, s \mapsto (s(v^1_\pi), \ldots, s(v^{d_1}_\pi))$ is the *input layer* function, where $v^j_\pi \in \mathcal{V}$ for $j \in \{1, \ldots, d_1\}$ denotes the state variable associated with input neuron j.
- $f_i \colon \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}, V \mapsto ReLU(W_i \cdot V + B_i)$, for $i \in \{2, \ldots, d-1\}$, is the function of *hidden layer* i. W_i is the weight matrix of layer i, i.e., $(W_i)_{j,k}$ denotes the weight of the output of neuron k in layer $i-1$ to the input of neuron j in layer i. B_i is the bias vector, i.e., $(B_i)_j$ denotes the bias of neuron j in layer i.
- $f_d \colon \mathbb{R}^{d_{d-1}} \to \mathbb{R}^{d_d}, V \mapsto W_d \cdot V + B_d$ is the function of output layer d. Here, no ReLU activation is applied.

We distinguish two variants of *neural action policies* implemented by f_π. The *no-filter* neural action policy implemented by f_π is the function $\pi_{napp} \colon \mathcal{S} \to \mathcal{L}, s \mapsto \operatorname{argmax}_{l \in \mathcal{L} \setminus \{\tau\}} f^l_\pi(s)$, where f^l_π denotes the output of f_π associated with l. The *app-filter* neural action policy implemented by f_π is the function $\pi_{app} \colon \mathcal{S} \to \mathcal{L}$,

$$s \mapsto \begin{cases} \underset{\{l \in \mathcal{L} \setminus \{\tau\} \mid \exists o \in \mathcal{O}_l \colon s \models g(o)\}}{\operatorname{argmax}} f^l_\pi(s) & \exists o \in \mathcal{O} \colon l(o) \neq \tau \\ & \wedge\, s \models g(o) \\ \tau & otherwise \end{cases}$$

where $\mathcal{O}_l = \{o \in \mathcal{O} \mid l(o) = l\}$ and τ is a special label, defined to be selected (by π_{app}) iff there does not exist a label $l \in \mathcal{L} \setminus \{\tau\}$ that is applicable in s.

SMT Encodings

We provide a specification of the various SMT checks deployed during PPA computation as well as CeAna. The encodings are largely compositional, and hence presented in a modular manner.

Encodings without π. Given a copy V of state variables $\mathcal{V}$, we denote the encoding of a constraint $\phi \in C(\mathcal{V})$ over V by $\Phi(\phi, V)$. The *the state variable bound encoding* $\Phi(V)$ is the conjunction $\bigcup_{v \in V} lo(v) \leq v \wedge v \leq up(v)$, where $lo(v)$

and $up(v)$ denote the lower and upper bound of the corresponding state variable respectively. The *abstract state encoding* of $s_{\mathcal{P}} \in \mathcal{S}_{\mathcal{P}}$, denoted $\Phi(s_{\mathcal{P}}, V)$, is the conjunction $\bigcup_{p \in s_{\mathcal{P}}} \begin{cases} \Phi(p, V) & s_{\mathcal{P}}(p) = 1 \\ \neg\Phi(p, V) & s_{\mathcal{P}}(p) = 0. \end{cases}$

For update $u \in U$ let V^u denote a u-indexed copy of V. The *update encoding* of u, denoted $\Phi(u, V, V^u)$, is the conjunction $\bigwedge_{v \in V} v^u = \Phi(u(v), V)$, where $v^u \in V^u$ denotes the u-copy of v. The *probabilistic update encoding* of $\bar{u}$, denoted $\Phi(\bar{u}, V)$, is the conjunction $\bigwedge_{u \in Supp(\bar{u})} \Phi(u, V, V^u)$. The *abstract distribution encoding* of $\mu_{\mathcal{P}} \in Dist(U \times \mathcal{S}_{\mathcal{P}})$, denoted $\Phi(\mu_{\mathcal{P}}, V)$, is the conjunction $\bigwedge_{(u, s'_{\mathcal{P}}) \in Supp(\mu_{\mathcal{P}})} \Phi(s'_{\mathcal{P}}, V^u)$.

The (no-π) ***abstract transition problem encoding*** of $(s_{\mathcal{P}}, o, \mu_{\mathcal{P}}) \in \mathcal{S}_{\mathcal{P}} \times \mathcal{O} \times Dist(U \times \mathcal{S}_{\mathcal{P}})$, denoted $\Phi((s_{\mathcal{P}}, o, \mu_{\mathcal{P}}), \mathcal{T}_{\mathcal{P}}, V)$ conjoins: $\Phi(V)$, $\bigwedge_{u \in Supp(\bar{u}(o))} \Phi(V^u)$, $\Phi(s_{\mathcal{P}}, V)$, $\Phi(g(o), V)$, $\Phi(\bar{u}(o), V)$, $\Phi(\mu_{\mathcal{P}}, V)$. We assume, w.l.o.g., that $\bar{u}(o)$ and $\mu_{\mathcal{P}}$ agree on U, i.e., $(u, s'_{\mathcal{P}}) \in Supp(\mu_{\mathcal{P}})$ only if $u \in Supp(\bar{u}(o))$, and for each $u \in Supp(\bar{u}(o))$ there exists exactly one $s'_{\mathcal{P}} \in \mathcal{S}_{\mathcal{P}}$ such that $(u, s'_{\mathcal{P}}) \in Supp(\mu_{\mathcal{P}})$ with $\mu_{\mathcal{P}}(u, s'_{\mathcal{P}}) = \bar{u}(o)(u)$.

For index i let V^i denote a i-indexed copy of V. The ***$\mathcal{T}$-path realization encoding*** **(i-a)** of abstract unsafe path $\sigma_{\mathcal{P}} \in Path(\Theta_{\mathcal{P}}, \phi_u)$, denoted $\Phi(\sigma_{\mathcal{P}}, \Theta, V)$ conjoins: $\bigwedge_{0 \le i \le |\sigma_{\mathcal{P}}|} \Phi(V^i)$, $\Phi(\phi_0, V^0)$, $\bigwedge_{0 \le i \le |\sigma_{\mathcal{P}}|} \Phi(s^i_{\mathcal{P}}(\sigma_{\mathcal{P}}), V^i)$, $\bigwedge_{0 \le i < |\sigma_{\mathcal{P}}|} \Phi(g^i(\sigma_{\mathcal{P}}), V^i) \wedge \Phi(u^i(\sigma_{\mathcal{P}}), V^i, V^{i+1})$, $\Phi(\phi_u, V^{|\sigma_{\mathcal{P}}|})$.

Note that (1) $\Phi(\neg\phi_u, V^i)$ for $0 \le i < |\sigma_{\mathcal{P}}|$ is entailed by over-approximation of $\sigma_{\mathcal{P}}$ and hence *not* explicitly encoded; (2) $\Phi(\sigma_{\mathcal{P}}, \Theta, V)$ – and thereby all compositional encodings thereof – is *distribution-insensitive* in that it constrains $s^i_{\mathcal{P}}$ but not $\mu^i_{\mathcal{P}}$. A solution to this encoding is a concrete path σ such that $|\sigma| = |\sigma_{\mathcal{P}}|$ and $o^i(\sigma) = o^i(\sigma_{\mathcal{P}})$, $u^i(\sigma) = u^i(\sigma_{\mathcal{P}})$, $s^i(\sigma) \in [s^{\sigma_{\mathcal{P}}}_{\mathcal{P}}]$ – but not necessarily $\mu^i(\sigma) \in [\mu^i_{\mathcal{P}}(\sigma_{\mathcal{P}})]$ – for $i \in \{0, \ldots, |\sigma|\}$. This is a commonplace enhancement (Hermanns et al. 2008), which allows off-the-shelf checking and refining of path spuriousness without adaptions for spuriousness induced by $\mu^i_{\mathcal{P}}$. It preserves correctness in that when counterexample analysis derives *realizable*, there exists $ups \subseteq Path(A, s^0, \phi_u)$ such that $Pr(ups) > p_u$ for *some* adversary A but not necessarily concretization $[A_{\mathcal{P}}]$.

For abstract path $\sigma_{\mathcal{P}}$ let $b_{\sigma_{\mathcal{P}}}$ denote a fresh binary variable. The ***$\mathcal{T}$-probabilistic maximal realization encoding*** **(ii-a)** of abstract unsafe path set $ups_{\mathcal{P}} \subseteq Path(\Theta_{\mathcal{P}}, \phi_u)$, denoted $\Phi_{max}(ups_{\mathcal{P}}, \Theta, V)$ has **maximization objective** $\sum_{\sigma_{\mathcal{P}} \in ups_{\mathcal{P}}} Pr(\sigma_{\mathcal{P}}) \cdot b_{\sigma_{\mathcal{P}}}$, and conjoins

$$\bigwedge_{\sigma_{\mathcal{P}} \in ups_{\mathcal{P}}} \left(\neg b_{\sigma_{\mathcal{P}}} \vee (\Phi(\sigma_{\mathcal{P}}, \Theta, V^{\sigma_{\mathcal{P}}}) \wedge \bigwedge_{v \in V} v = v^{\sigma_{\mathcal{P}}, 0}) \right).$$

$\Phi(ups_{\mathcal{P}}, \Theta, V)$ maximizes the probability mass of paths that are realizable from some common concrete start state s^0_* encoded over V. Intuitively, $b_{\sigma_{\mathcal{P}}}$ constrains that, only if $\sigma_{\mathcal{P}}$ is realizable from s^0_*, it contributes to the maximization.

Encodings with π. Given a copy V of state variables V and a neural network f_π, x^i_j and z^i_j denote *real-valued auxiliary variables* of neuron (i, j), which are implicitly V-indexed. The *NN encoding of* neural network f_π, denoted $\Phi(f_\pi, V)$, conjoins: $\bigwedge_{1 \le j \le d_1} x^1_j = v^j_\pi$, $\bigwedge_{2 \le i \le d-1} \bigwedge_{1 \le j \le d_i} z^i_j = \sum_{k=1}^{d_{i-1}} (W_i)_{j,k} \cdot x^{i-1}_k + (B_i)_j$, $\bigwedge_{2 \le i \le d-1} \bigwedge_{1 \le j \le d_i} x^i_j = ReLU(z^i_j)$, and $\bigwedge_{1 \le j \le d_d} x^d_j = \sum_{k=1}^{d_{d-1}} (W_d)_{j,k} \cdot x^{d-1}_k + (B_d)_j$. Note that this encoding is solver specific in that it assumes a special construct for ReLU constraints (Katz et al. 2019).

Let $l \in \mathcal{L} \setminus \{\tau\}$. Let x^d_l denote the auxiliary variable of the output neuron associated with label l. The *policy selection encoding* of π and l, denoted $\Phi(\pi, l, V)$, conjoins $\Phi(f_\pi, V)$ and $\bigwedge_{l' \in \mathcal{L} \setminus \{l, \tau\}} x^d_l > x^d_{l'}$ (if π is a no-filter policy) or $\bigwedge_{l' \in \mathcal{L} \setminus \{l, \tau\}} \left(x^d_l > x^d_{l'} \vee \neg \bigvee_{o \in \mathcal{O}_{l'}} \Phi(g(o), V) \right)$ (if π is an app-filter policy).

The ***π-abstract transition problem encoding*** of $(s_{\mathcal{P}}, o, \mu_{\mathcal{P}}) \in \mathcal{S}_{\mathcal{P}} \times \mathcal{O} \times Dist(U \times \mathcal{S}_{\mathcal{P}})$, denoted $\Phi((s_{\mathcal{P}}, o, \mu_{\mathcal{P}}), \mathcal{T}^\pi_{\mathcal{P}}, V)$, is the conjunction $\Phi((s_{\mathcal{P}}, o, \mu_{\mathcal{P}}), \mathcal{T}_{\mathcal{P}}, V) \wedge \Phi(\pi, l(o), V)$.

The ***π-path realization encoding*** **(i-b)** of abstract unsafe path $\sigma_{\mathcal{P}} \in Path(\Theta^\pi_{\mathcal{P}}, \phi_u)$, denoted $\Phi(\sigma_{\mathcal{P}}, \Theta^\pi, V)$ is the conjunction $\Phi(\sigma_{\mathcal{P}}, \Theta, V) \wedge \bigwedge_{0 \le i < |\sigma_{\mathcal{P}}|} \Phi(\pi, l(o^i(\sigma_{\mathcal{P}})), V^i)$.

The ***π-all realization encoding*** **(ii-b)** of abstract unsafe path set $ups_{\mathcal{P}} \subseteq Path(\Theta_{\mathcal{P}}, \phi_u)$, denoted $\Phi(ups_{\mathcal{P}}, \Theta^\pi, V)$ conjoins $\bigwedge_{\sigma_{\mathcal{P}} \in ups_{\mathcal{P}}} \Phi(\sigma_{\mathcal{P}}, \Theta^\pi, V^{\sigma_{\mathcal{P}}}) \wedge \bigwedge_{v \in V} v = v^{\sigma_{\mathcal{P}}, 0}$.

Automata Network

Vea's code base supports PTS encoded in the automata language JANI (Budde et al. 2017). We now show the automata network structure underlying the generic PTS description $\langle \mathcal{V}, \mathcal{L}, \mathcal{O} \rangle$ in the main text.

A *network of automata* is a tuple $\langle \mathsf{V}, \mathsf{L}, \mathsf{A}, \mathsf{Sync} \rangle$, where

- V is a finite set of variables, each with a bounded integer domain.
- L is a finite set of labels, excluding the *silent label* $\tau \notin \mathsf{L}$.
- A is a finite set of automata.
- $\mathsf{Sync} \subseteq (\mathsf{A} \to \mathsf{L}) \times (\mathsf{L} \cup \{\tau\})$ is a finite set of *synchronization* constraints.

An *automaton* a is a tuple $\langle \mathsf{Loc}, \mathsf{E} \rangle$, where Loc is a nonempty finite set of *locations*, and E is a finite set of *edges* of a. An *edge* e of a is a tuple $(\mathsf{l}, \mathsf{loc_s}, \mathsf{g}, \bar{\mathsf{u}}) \in \mathsf{E}$ with

- label $\mathsf{l} \in \mathsf{L}$ for labeled edges or $\mathsf{l} = \tau$ for silent edges.
- source location $\mathsf{loc_s} \in \mathsf{Loc}$.
- guard $\mathsf{g} \in C(\mathsf{V})$.

- probabilistic update $\bar{\mathsf{u}} \in \mathit{Dist}(\mathsf{Loc} \times (\mathsf{V} \to \mathit{Exp}(\mathsf{V})))$, where each $(\mathsf{loc_d}, \mathsf{u}) \in \mathit{Supp}(\bar{\mathsf{u}})$ is composed of a destination location $\mathsf{loc_d}$ and a partial variable update u with $\mathit{dom}(\mathsf{u}) \subseteq \mathsf{V}$.

An automaton consists of a set of *locations* connected by *edges*. Each edge links from a *source* location to finitely many *destination* locations, each weighted with some non-zero probability. An edge can be taken, i.e., the automaton can transit from the source to some destination of the edge, only if its *guard* evaluates to true over the current variable assignment. If an edge is taken, the variables are updated according to the *update* associated with the destination location. While *silent* edges can be taken independently, *labeled* edges can only be taken as part of a *synchronization*. Here, a synchronization constraint specifies for each automaton – possibly from a subset of participating automata – an action label. Additionally, it specifies the label of the synchronization. Under this label, the participating automata may synchronize taking edges whose label combination agrees with the synchronization constraint.

The state space description $\langle \mathcal{V}, \mathcal{L}, \mathcal{O} \rangle$ of an automata network $\langle \mathsf{V}, \mathsf{L}, \mathsf{A}, \mathsf{Sync} \rangle$ is obtained as follows:

- $\mathcal{V} = \mathsf{V} \cup \{v_\mathsf{a} \mid \mathsf{a} \in \mathsf{A}\}$, where v_a is the location variable of automaton a. $D(v_\mathsf{a}) = \mathsf{Loc}(\mathsf{a})$ is interpreted as a bounded-integer interval.
- $\mathcal{L} = \mathsf{L} \cup \{\tau\}$.
- $\mathcal{O}$ contains an operator $(l, g, \bar{u})$
  - for each silent edge $(\tau, \mathsf{loc_s}, \mathsf{g}, \mathsf{u}, \mathsf{loc_d}) \in \mathsf{E}(\mathsf{a})$ in each automaton $\mathsf{a} \in \mathsf{A}$ where $g := v_\mathsf{a} = \mathsf{loc_s} \wedge \mathsf{g}$, $l := \tau$, and $u \in \mathit{Supp}(\bar{u})$ for each $(\mathsf{loc_d}, \mathsf{u}) \in \mathit{Supp}(\bar{\mathsf{u}})$ with $u = \mathsf{u} \cup \{v_\mathsf{a} \mapsto \mathsf{loc_d}\} \cup \{v \mapsto v \mid v \in \mathcal{V} \setminus (\mathit{dom}(\mathsf{u}) \cup \{\mathsf{a}\})\}$ and $\bar{u}(u) = \bar{\mathsf{u}}(\mathsf{u})$,
  - for each synchronization constraint $(\lambda, \mathsf{l}) \in \mathsf{Sync}$ with $\mathit{dom}(\lambda) = \{\mathsf{a}^1, \ldots, \mathsf{a}^n\}$ and each combination of edges $\mathsf{e}^1 \in \mathsf{E}(\mathsf{a}^1), \ldots, \mathsf{e}^n \in \mathsf{E}(\mathsf{a}^n)$ such that $\mathsf{e}^i = (\lambda(\mathsf{a}^i), \mathsf{loc}_\mathsf{s}^i, \mathsf{g}^i, \bar{\mathsf{u}}^i)$ for $i \in \{1, \ldots, n\}$, where $l = \mathsf{l}$, $g := \wedge_{i=1}^n (v_{\mathsf{a}^i} = \mathsf{loc}_\mathsf{s}^i \wedge \mathsf{g}^i)$, and $u \in \mathit{Supp}(\bar{u})$ for each combination $\mathsf{u}^1 \in \mathit{Supp}(\bar{u}^1), \ldots, \mathsf{u}^n \in \mathit{Supp}(\bar{u}^n)$ with $u = \left(\bigcup_{i=1}^n \mathsf{u}^i \cup \{v_{\mathsf{a}^i} \mapsto \mathsf{loc}_\mathsf{d}^i\}\right) \cup \{v \mapsto v \mid v \in \mathcal{V} \setminus \bigcup_{i=1}^n (\mathit{dom}(\mathsf{u}^i) \cup \{v_{\mathsf{a}^i}\})\}$ and $\bar{u}(u) = \prod_{i=1}^n \bar{\mathsf{u}}(\mathsf{u})$.

The silent label τ models edges and synchronizations that can be taken independent of a control policy π. That is, $\mathcal{T}^\pi = \{(s, o, \mu) \in \mathcal{T} \mid \pi(s) = l(o) \vee l(o) = \tau\}$. We silently ignore this special case in our approach. Its implementation, e.g., for abstract state expansion or realization checks, is straight forward.

Encoding the Policy Into the Automata Network

In our approach, the neural policy π is a distinctive component of the safety problem. Alternatively, the composition of π and π-controlled PTS (modeled as an automata network), can again be encoded as an automata network. This enables us to feed the neural policy safety analysis into off-the-shelf probabilistic model checkers.

Many SOA probabilistic model checkers, e.g., STORM, focus on discrete state systems in that they do not fully support real-valued state variables. We thus provide two distinct encodings of the neural policy, one with *real-valued variables* and another that *discretizes* the NN structure with finite precision n based on *rational arithmetic*.

Encoding with real-valued variables. Let π be a neural action policy, and let $\mathsf{n} = \langle \mathsf{V}, \mathsf{L}, \mathsf{A}, \mathsf{Sync} \rangle$ be the π-controlled automata network. The $(\mathsf{n} \mid \pi)$-*composition* is the automata network $\langle \mathsf{V} \cup \mathsf{V}_\pi, \mathsf{L}, \mathsf{A} \cup \{\mathsf{a}_\pi\}, \mathsf{Sync}_\pi \rangle$ where

- V_π is the set of *neural policy variables*. $\mathsf{x}_j \in \mathsf{V}_\pi$ for each neuron index $j \in 1, \ldots, d_i$ over the **even-index** layers $i \in 2, \ldots, d$, where $D(\mathsf{x}_j)$ is the **least tight** interval as per interval arithmetic (Moore et al. 2009) over all corresponding neurons (i, j). $\mathsf{y}_j \in \mathsf{V}_\pi$ for each neuron index $j \in 1, \ldots, d_i$ over the **odd-index** layers $i \in 3, \ldots, d$, with $D(\mathsf{y}_j)$ analogously to even-index variables. If π is **app-filter**, then $\mathsf{v_l} \in \mathsf{V}_\pi$ for each $\mathsf{l} \in \mathsf{L}$, where $D(\mathsf{v_l}) = \{0, 1\}$.
- $\mathsf{a}_\pi = \langle \mathsf{Loc}_\pi, \mathsf{E}_\pi \rangle$ is the *policy automaton*.
- $\mathsf{Sync}_\pi = \{(\lambda \cup \{\mathsf{a}_\pi \mapsto \mathsf{l}\}, \mathsf{l}) \mid (\lambda, \mathsf{l}) \in \mathsf{Sync} \wedge \mathsf{l} \neq \tau\} \cup \{(\lambda, \tau) \in \mathsf{Sync}\}$ is the *policy-annotated synchronization set*.

The policy automaton $\mathsf{a}_\pi = \langle \mathsf{Loc}_\pi, \mathsf{E}_\pi \rangle$ is composed of locations $\mathsf{Loc}_\pi = \{\mathsf{loc}_i \mid i \in \{1, \ldots, d\}\}$ and edges E_π,

- for each layer $1 < i \leq d - 1$ a deterministic *to-hidden-layer* edge $(\tau, \mathsf{loc}_{i-1}, 1, \bar{\mathsf{u}}) \in \mathsf{E}_\pi$ where $\mathit{Supp}(\bar{\mathsf{u}}) = \{(\mathsf{loc}_i, \mathsf{u})\}$ and $\mathsf{u}(\mathsf{z}_j^i) = \mathit{ReLU}(\sum_k^{d_{i-1}} (W_i)_{j,k} \cdot \mathsf{z}_k^{i-1} + (B_i)_j)$ for each j in $1, \ldots, d_i$. z_j^i denotes v_π^j for the input layer $i = 1$, x_j for even layers i and y_j for odd layers $i > 1$.
- a deterministic *to-output-layer* edge $(\tau, \mathsf{loc}_{d-1}, 1, \bar{\mathsf{u}}) \in \mathsf{E}_\pi$ where $\mathit{Supp}(\bar{\mathsf{u}}) = \{(\mathsf{loc}_i, \mathsf{u})\}$ and $\mathsf{u}(\mathsf{z}_j^d) = \sum_k^{d_{d-1}} (W_d)_{j,k} \cdot \mathsf{z}_k^{d-1} + (B_d)_j$ for each j in $1, \ldots, d_d$, and if π is **app-filter**, then $u(\mathsf{v_l}) = \bigvee_{o \in \mathcal{O}_\mathsf{l}} g(o)$ for each $\mathsf{l} \in \mathsf{L}$ with $\mathcal{O}_l$ as induced by n.
- for each $\mathsf{l} \in \mathsf{L}$ a *synchronization* edge $(\mathsf{l}, \mathsf{loc}_d, \mathsf{g}, \bar{\mathsf{u}}) \in \mathsf{E}_\pi$ with $\mathit{Supp}(\bar{\mathsf{u}}) = \{(\mathsf{loc}_1, \mathsf{u})\}$, $\mathsf{u} = \emptyset$ and $\mathsf{g} = \bigwedge_{\mathsf{l}' \in \mathsf{L} \setminus \{\mathsf{l}\}} \mathsf{z_l} > \mathsf{z_{l'}}$ for **no-filter** and alternatively $\mathsf{g} = \bigwedge_{\mathsf{l}' \in \mathsf{L} \setminus \{\mathsf{l}\}} \mathsf{z_l} > \mathsf{z_{l'}} \vee \neg \mathsf{v_{l'}}$ for **app-filter**, where $\mathsf{z_l}$ denotes the encoding variable of the output neuron associated with l.

The encoding distinguishes between neural policies *with and without applicability filter*. π is encoded as an additional automaton in the network. Labeled transitions are controlled by π via synchronization constraints. We introduce additional state variables to encode the NN output computation.[1] The layer-wise structure allows to introduce two sets of real-valued variables only, one for even-index layers and one for odd-index layers, rather than one variable per neuron in the network. Given the domain bounds of the state

[1]A compact symbolic encoding of the NN output, i.e., recursively inlining the layer structure, results is an infeasible blow up in encoding size due to the piecewise-linear ReLU activations.

variable inputs, interval arithmetic (Moore et al. 2009) can be applied to derive bounds on each neuron in the NN. The domain of z_j^i is then the least tight interval, i.e., the smallest lower bound and the largest upper bound over all corresponding neurons. Alternatively, if supported by the model checker, e.g., ePMC and MODEST, the domain can be unbounded. The NN output computation is encoded by the silent and deterministic *to-hidden-layer* and *to-output-layer* edges. $ReLU(\mathsf{z})$ is syntactic sugar for $\max(\mathsf{z}, 0)$. Alternatively, one could also use an *if-then-else* construct. If strictly limited to linear expressions, the piecewise-linear case distinction ($\mathsf{z} > 0$) could be encoded via two distinct edges.

For applicability-filtering, we introduce an additional set of binary applicability flags. These are set by the to-output-layer edge. In the start constraint ϕ_0 of a safety property, each variable $\mathsf{v} \in \mathsf{V}_\pi$ is fixed to an arbitrary value. The initial location is loc_0.

Discretized encoding with rational arithmetic. The discretized encoding has finite precision $n \in \mathbb{N}$ and uses rational arithmetic. It maintains a unified denominator $q = 10^n$. NN weights and biases, in floating representation, are approximated with precision n by a rational number with denominator q. For instance, let $n = 4$, then $2.12341\dots$ is approximated by $\frac{21234}{10000}$. The denominator is maintained implicitly.[2]

The neuron policy variables $\mathsf{z} \in \mathsf{V}_\pi$ encode the numerators of the underlying rational values. $D(\mathsf{z})$ is an integer interval. The lower and upper bound is the numerator of the *floor* and *ceil* value of the real-valued interval bounds transformed to denominator q respectively.

The encoding of to-hidden-layer edges for layer $i < d-1$ and neuron j is adapted from the real-valued encoding as follows

$$\mathsf{u}(\mathsf{z}_j^i) = ReLU(\sum_k^{d_{i-1}} \left\lfloor \frac{num((W_i)_{j,k}) \cdot \mathsf{z}_k^{i-1}}{q} \right\rfloor + num((B_i)_j))$$

where $num(w)$ denotes the numerator of the rational number approximation of w with precision n. The encoding of the to-output-layer edge is adapted accordingly. The division by q preservers the unified denominator in that $ReLU(\frac{p_1}{q} \cdot \frac{p_2}{q} + \frac{p_0}{q}) \equiv ReLU(\frac{\frac{p_1 \cdot p_2}{q}}{q} + \frac{p_0}{q}) \equiv ReLU(\frac{\frac{p_1 \cdot p_2}{q} + p_0}{q}) \equiv \frac{ReLU(\frac{p_1 \cdot p_2}{q} + p_0)}{q}$. Note that, due to the division, the update encoding is non-linear. That said, the SOA model checkers we deploy all support this language fragment. The encoding of the synchronization edge remains syntactically unmodified, with respect to the real-valued encoding, since $\frac{z}{q} > \frac{z'}{q} \equiv z > z'$.

Due to the finite precision approximation on the encoding level, the discretization does not guarantee to mimic the true policy-restricted system faithfully. Imprecision may accumulate over systems executions. Given the checkers' limitations, this is the best possible basis for a performance comparison. In particular, the size of the NN-controlled system is equal across encodings.

In our evaluation, we experiment with $n = 3, \dots, 7$. We obtain similar results independent of n: All configuration exceed either time or memory limits on all problem instances. The only exception is STORM's sparse engine which successfully terminates on 6 Blocks (CI) NN 16 (without app-filter) after 150 to 180 seconds for $n \leq 5$. In the main text, we report the runtime for $n = 5$.

JANI-to-PRISM translation. The PRISM and the PET support input models in PRISM's own input language only. We translate automata networks encoded in JANI to PRISM in a straight-forward manner. Each (JANI) automaton becomes a (PRISM) *module*, with the automaton location encoded as an additional bounded-integer module variable. In PRISM all modules that share a common label must participate in the synchronization under this label. While this is a strict subset of the synchronization constraints supported in JANI, it is sufficient for our purposes (on our benchmarks).

Start state enumeration. The probabilistic model checkers that we experiment with support sets of start states compactly represented by a constraint ϕ_0. The only exception is MODEST. Here, we encode *rejection-based* enumeration of ϕ_0 directly into the automata network. In a initialization step, the automata network non-deterministically assigns a value to each variable $\mathsf{v} \in \mathsf{V}$. If the resulting state s satisfies ϕ_0, the system execution proceeds from s. Otherwise s is terminal.

[2]Explicitly encoding the denominator via an additional variable for each neuron value variable is bound to overflow issues due to denominator unification for addition.

References

Bianco, A.; and de Alfaro, L. 1995. Model Checking of Probabalistic and Nondeterministic Systems. In *Foundations of Software Technology and Theoretical Computer Science, 15th Conference, Bangalore, India, December 18-20, 1995, Proceedings*, volume 1026 of *LNCS*, 499–513. Springer.

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.

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

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

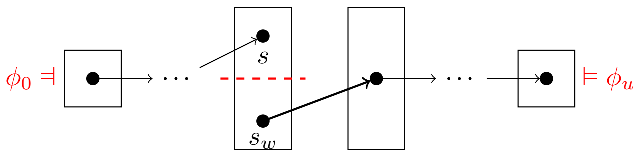
Katz, G.; Huang, D. A.; Ibeling, D.; Julian, K.; Lazarus, C.; Lim, R.; Shah, P.; Thakoor, S.; Wu, H.; Zeljic, A.; Dill, D. L.; Kochenderfer, M.; and Barrett, C. 2019. The Marabou Framework for Verification and Analysis of Deep Neural Networks. In *CAV*, volume 11561 of *LNCS*, 443–452. Springer.

Moore, R. E.; Kearfott, R. B.; and Cloud, M. J. 2009. *Introduction to Interval Analysis*. SIAM.

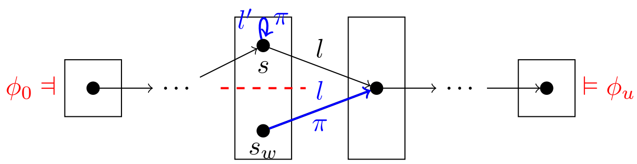
Vinzent, M.; Sharma, S.; and Hoffmann, J. 2023. Neural Policy Safety Verification via Predicate Abstraction: CEGAR. In *AAAI*, 15188–15196. AAAI Press.

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

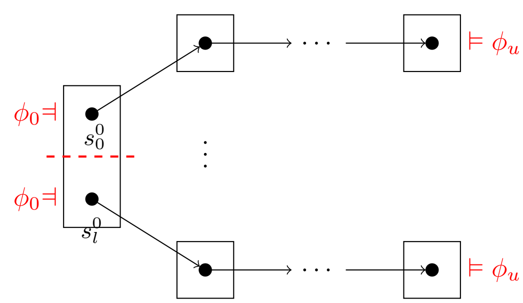
Wachter, B.; Zhang, L.; and Hermanns, H. 2007. Probabilistic Model Checking Modulo Theories. In *Fourth International Conference on the Quantitative Evaluaiton of Systems (QEST 2007), 17-19 September 2007, Edinburgh, Scotland UK*, 129–140. IEEE Computer Society.



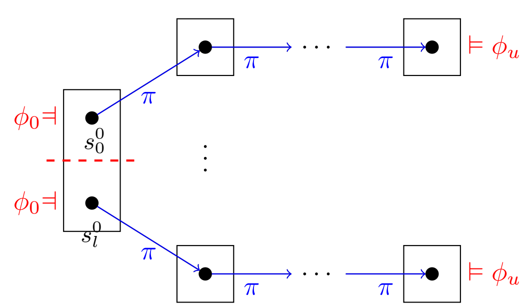
(a) (i-a) $\mathcal{T}$-path spuriousness: Path is not realizable in Θ.



(b) (i-b) π-path spuriousness: Path is realizable in Θ but **not** in Θ^π.



(c) (ii-a) $\mathcal{T}$-probabilistic spuriousness: Individual paths of $ups_\mathcal{P}$ are realizable in Θ but **not** from a common start state $s^0 \models \phi_0$.



(d) (ii-b) π-probabilistic spuriousness: $ups_\mathcal{P}$ is realizable in Θ and individual paths of $ups_\mathcal{P}$ are realizable in Θ^π, but they are **not** realizable from a common start state $s^0 \models \phi_0$ in Θ^π.

Figure 5: Sources of spuriousness in prob-CEGAR-PPA.

Algorithm 3: Probabilistic abstract state expansion.

Input: $s_\mathcal{P} \in \mathcal{S}_\mathcal{P}$.

for each $o \in \mathcal{O}$ **do**
  `// Applicability test:`
  if $\neg(s_\mathcal{P} \models g(o))$ **then continue**
  for each $\mu_\mathcal{P} \in$ `EnumPaDist` $(s_\mathcal{P}, o)$ **do**
    $g, \bar{u}, l \leftarrow g(o), \bar{u}(o), l(o)$
    `//` π`-transition test:`
    if $\exists s \in [s_\mathcal{P}]\colon s \models g \wedge s[\![\bar{u}]\!] \in [\mu_\mathcal{P}] \wedge \pi(s) = l$ **then**
      process $(s_\mathcal{P}, o, \mu_\mathcal{P})$

Procedure `EnumPaDist` $(s_\mathcal{P}, o)$**:**
  for each $u \in Supp(\bar{u}(o))$ **do**
    $S'_\mathcal{P}(u) \leftarrow$ `EnumPaSuc` $(s_\mathcal{P}, o, u)$
  $Dist_\mathcal{P}, \mu_\mathcal{P} \leftarrow \emptyset, \{(u, s'_\mathcal{P}) \mapsto 0 \mid (u, s_\mathcal{P}) \in U \times \mathcal{S}_\mathcal{P}\}$
  `EnumPaDist` $(Dist_\mathcal{P}, \mu_\mathcal{P})$
  return $Dist_\mathcal{P}$

Procedure `EnumPaDist` $(Dist_\mathcal{P}, \mu_\mathcal{P})$**:**
  if $Supp(\mu_\mathcal{P}) = Supp(\bar{u}(o))$ **then**
    `//` $\mathcal{T}$`-transition test:`
    if $\exists s \in [s_\mathcal{P}]\colon s \models g(o) \wedge s[\![\bar{u}(o)]\!] \in [\mu_\mathcal{P}]$ **then**
      $Dist_\mathcal{P} \leftarrow Dist_\mathcal{P} \cup \{\mu_\mathcal{P}\}$
    return
  Select $u \in Supp(\bar{u}(o)) \setminus Supp(\mu_\mathcal{P})$
  for each $s'_\mathcal{P} \in S'_\mathcal{P}(u)$ **do**
    $\mu'_\mathcal{P} \leftarrow \mu_\mathcal{P} \cup \{(u, s'_\mathcal{P}) \mapsto \bar{u}(o)(u)\}$
    `EnumPaDist` $(Dist_\mathcal{P}, \mu_\mathcal{P}')$

`// Adopted from Vea (2022):`
Procedure `EnumPaSuc` $(s_\mathcal{P}, o, u)$**:**
  $S'_\mathcal{P}, s'_\mathcal{P} \leftarrow \emptyset, \emptyset$
  `// Operator entailment:`
  for each $(p, b) \in \mathcal{P} \times \{0, 1\}$ **do**
    if $\forall s \in [s_\mathcal{P}] \cap [g(o)]\colon p(s[\![u]\!]) = b$ **then**
      $s'_\mathcal{P}(p) \leftarrow b$
  `EnumPaSuc` $(S'_\mathcal{P}, s'_\mathcal{P})$
  return $S'_\mathcal{P}$

Procedure `EnumPaSuc` $(S'_\mathcal{P}, s'_\mathcal{P})$**:**
  if $dom(s'_\mathcal{P}) = \mathcal{P}$ **then**
    `// Non-prob.` $\mathcal{T}$`-transition test:`
    if $\exists s \in [s_\mathcal{P}]\colon s \models g(o) \wedge s[\![u]\!] \in [s'_\mathcal{P}]$ **then**
      $S'_\mathcal{P} \leftarrow S'_\mathcal{P} \cup \{s'_\mathcal{P}\}$
    return
  Select $p \in \mathcal{P} \setminus dom(s'_\mathcal{P})$
  for each $b \in \{0, 1\}$ **do**
    $s''_\mathcal{P} \leftarrow s'_\mathcal{P} \cup \{p \mapsto b\}$
    `// Predicate entailment:`
    for each $(p', b') \in (\mathcal{P} \setminus dom(s''_\mathcal{P})) \times \{0, 1\}$ **do**
      if $\forall s \in \mathcal{S}\colon p(s) = b \rightarrow p'(s) = b'$ **then**
        $s''_\mathcal{P}(p') \leftarrow b'$
    `EnumPaSuc` $(S'_\mathcal{P}, s''_\mathcal{P})$