Formal Methods meet Neural Networks: A Selection

Tom Henzinger
IST Austria

Formal Methods

1. Static: ensure at design time that a system satisfies its specification

    1a. Verification: given system f and spec φ, does f satisfy φ?

    - deductive reasoning (logic, decision procedures)
    - algorithmic reasoning (model checking, abstract interpretation)

Formal Methods

1. Static: ensure at design time that a system satisfies its specification

   1a. Verification: given system f and spec φ, does f satisfy φ?

   - deductive reasoning (logic, decision procedures)
   - algorithmic reasoning (model checking, abstract interpretation)

   $\forall$traces

   1b. Synthesis: given spec φ, find system f such that f satisfies φ.

   - syntax-guided (translation, search, learning)
   - semantics-guided (game solving, control)

   $\forall$inputs $\exists$output

Formal Methods

1. Static: ensure at design time that a system satisfies its specification

    1a. Verification: given system f and spec φ, does f satisfy φ?

    - deductive reasoning (logic, decision procedures)
    - algorithmic reasoning (model checking, abstract interpretation)

    $\forall$traces

    1b. Synthesis: given spec φ, find system f such that f satisfies φ.

    - syntax-guided (translation, search, learning)
    - semantics-guided (game solving, control)

    $\forall$inputs $\exists$output

2. Dynamic: watch at runtime if a system satisfies its specification

    2a. Runtime monitoring
    2b. Runtime enforcement

Formal Methods meet Neural Networks: A Selection

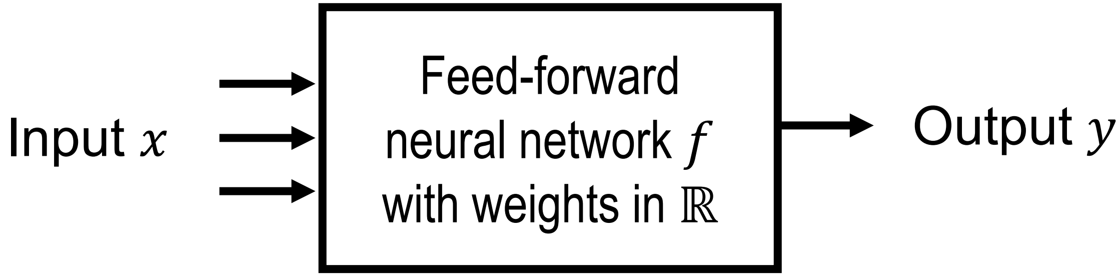
1. Open-loop verification: prove properties of neural networks
   [TACAS’20, AAAI’21] (joint work with Giacobbe-Lechner-Zikelic)

2. Closed-loop verification: prove properties of neural network controllers over discrete-time dynamical systems
   [NeurIPS’21, AAAI’22] (joint work with Lechner-Zikelic-Chatterjee)

3. Monitoring: runtime monitors as novelty detectors
   [ECAI’20, RV’21] (joint work with Lukina-Schilling)

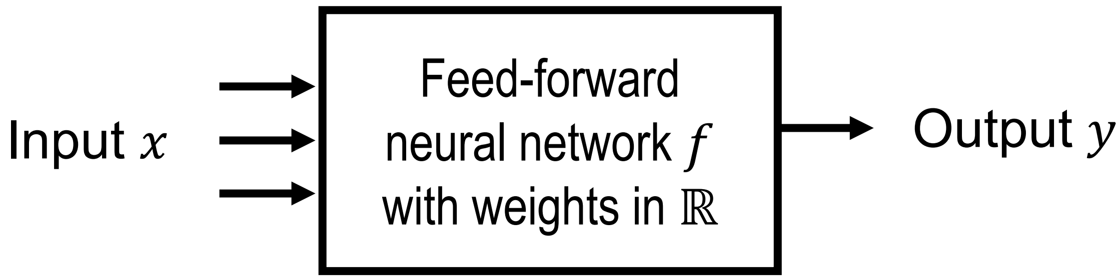
4. Synthesis/enforcement: runtime monitors as specification enforcers
   [CAV’19] (joint work with Avni-Bloem-Chatterjee-Koenighofer-Pranger)

Open-Loop Verification



$$\varphi(x) \wedge f(x) = y \Rightarrow \psi(y)$$

Open-Loop Verification



$$\varphi(x) \wedge f(x) = y \Rightarrow \psi(y)$$

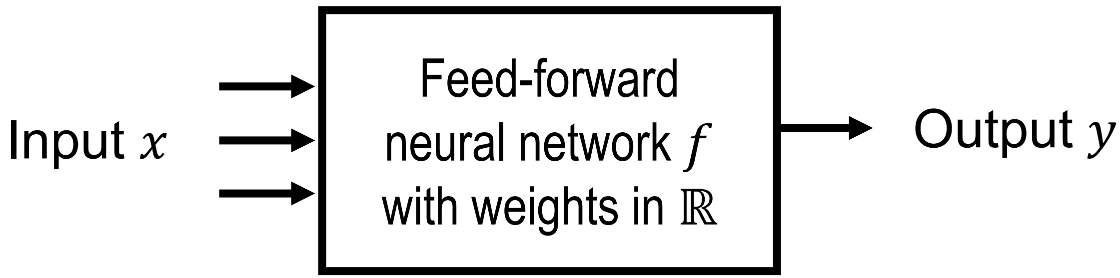
NP-complete [Katz'17]

$$\varphi(x) \wedge [f]_{\mathrm{float32}}(x) = y \Rightarrow \psi(y)$$

$$\varphi(x) \wedge [f]_{\mathrm{int8}}(x) = y \Rightarrow \psi(y)$$

PSPACE-hard [AAAI'21]

Open-Loop Verification



1. Abstraction domains over $\mathbb{R}^n$ (intervals, zonotopes, polyhedra)
2. Constraint solvers (MILP, Reluplex, NRA)

$$\varphi(x) \land f(x) = y \Rightarrow \psi(y)$$

[Ehlers, Katz et al., Bunel et al., Dutta et al., Mirman et al., Wang et al., Chang et al., Tjeng et al., etc.]

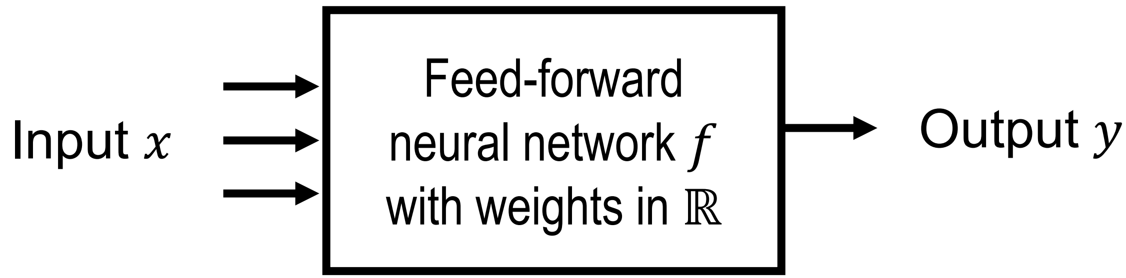
$$\varphi(x) \land [f]_{\text{float32}}(x) = y \Rightarrow \psi(y)$$

Bitvector SMT

$$\varphi(x) \land [f]_{\text{int8}}(x) = y \Rightarrow \psi(y)$$

[Baranowski et al., TACAS'20, AAAI'21]

Open-Loop Verification

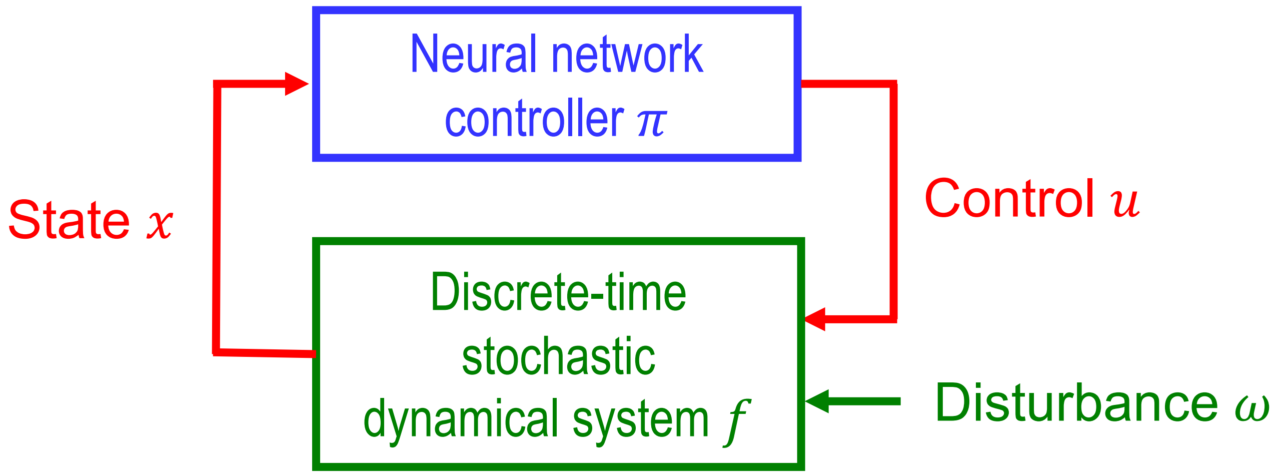


ε-robustness at input x:

$$(\forall x')\left(d(x,x') < \varepsilon \;\Rightarrow\; f(x) = f(x')\right)$$

Robustness is not preserved by $[\cdot]_{\mathrm{float}k}$ nor by $[\cdot]_{\mathrm{int}k}$ nor is robustness monotonic in k [TACAS'20, Jia-Rinard].

Closed-Loop Verification



Safety: unsafe state set X_U is never entered (infinite time horizon!).

Progress (stability): stable state set X_S is eventually entered with probability 1 ("almost surely") and never left.

Discrete-Time Stochastic Dynamical System

State space	$X \subseteq \mathbb{R}^n$
Control policy	$\pi: X \to \mathcal{D}(U)$
Disturbance	$\omega \sim \mathcal{D}(W)$
Initial state	$x_0 \in X$
Transition function	$f: X \times U \times W \to X$

Deterministic control policy: given by neural network.

Stochastic control policy: by Bayesian neural network (BNN).

Safety Certificates: Inductive Invariants

Inductive (“positive”) invariant:

continuous function $S: X \to \mathbb{R}$ such that

(1) $S(x_0) > 0$
(2) for all $x \in X, u \sim \pi(x)$, and ω, if $S(x) > 0$ then $S\big(f(x, u, \omega)\big) > 0$
(3) for all unsafe states $x \in X_U$, $S(x) \leq 0$

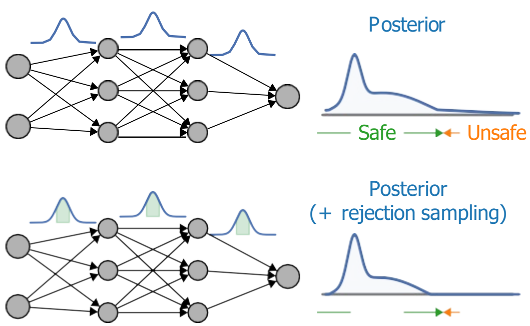
If there exists an inductive invariant,
then the closed-loop system is safe.

Safety Certificates: Inductive Invariants

Inductive ("positive") invariant:

continuous function $S: X \to \mathbb{R}$ such that

(1) $S(x_0) > 0$
(2) for all $x \in X, u \sim \pi(x)$, and ω, if $S(x) > 0$ then $S\big(f(x,u,\omega)\big) > 0$
(3) for all unsafe states $x \in X_U$, $S(x) \leq 0$



If there exists an inductive invariant, then the closed-loop system is safe.

BNNs are usually not safe (because of weight distributions with unbounded support), but can be made safe by rejection sampling (cut off support at $\mu \pm \delta$ for mean weight μ).

Progress Certificates: Ranking Supermartingales

Ranking supermartingale [Chakarov-Sankaranarayanan]:

continuous function $P: X \to \mathbb{R}$ such that

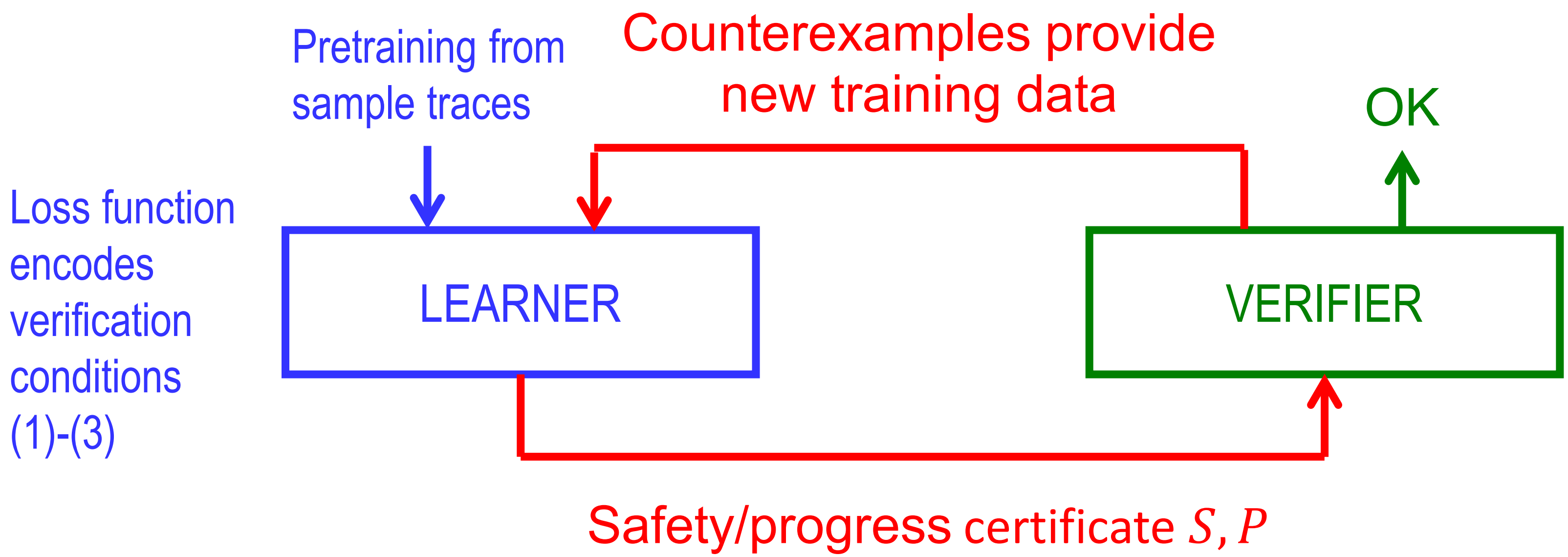
(1) for all $x \in X$, $S(x) \geq 0$
(2) there exists $\varepsilon > 0$ such that for all unstable $x \in X \backslash X_S, u \sim \pi(x)$, and ω,

$$\mathbb{E}_{u,\omega}\left[P\big(f(x,u,\omega)\big)\right] \leq P(x) - \varepsilon$$

(3) for all stable $x \in X_S$, $u \sim \pi(x)$, and ω, $f(x,u,\omega) \in X_S$

If there exists a ranking supermartingale,
then the closed-loop system is almost surely stable.
Moreover, we can bound the stabilization time [AAAI'22].

Learning and Verifying Certificates

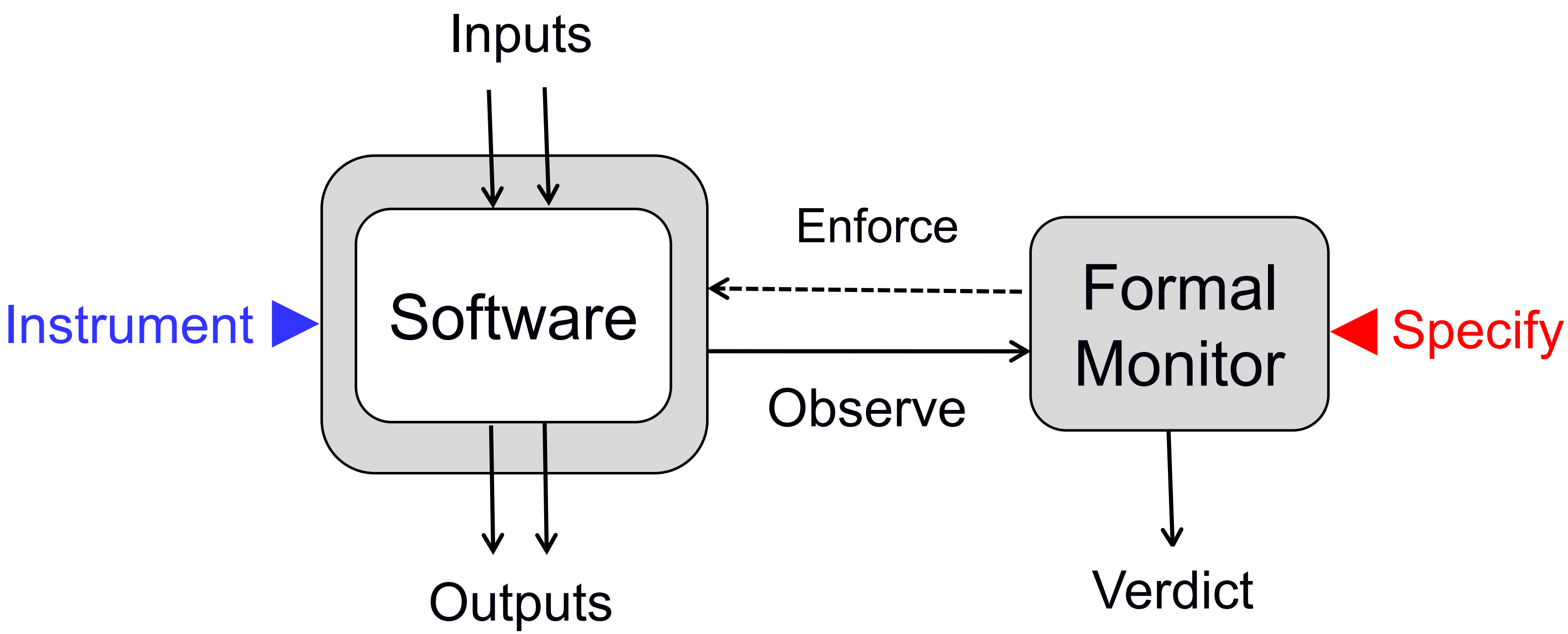


Constraint solvers:
-Mixed integer linear programming
-Reluplex [Katz et al.]
-Nonlinear real arithmetic

Software Needs Watchdogs

Tom Henzinger
IST Austria

The Vision:
Ubiquitous Online Black-Box Monitoring of Software



Bounded number of monitor operations per observation (“real time” [Rabin’63]).

Two Megatrends that Favor Runtime Verification

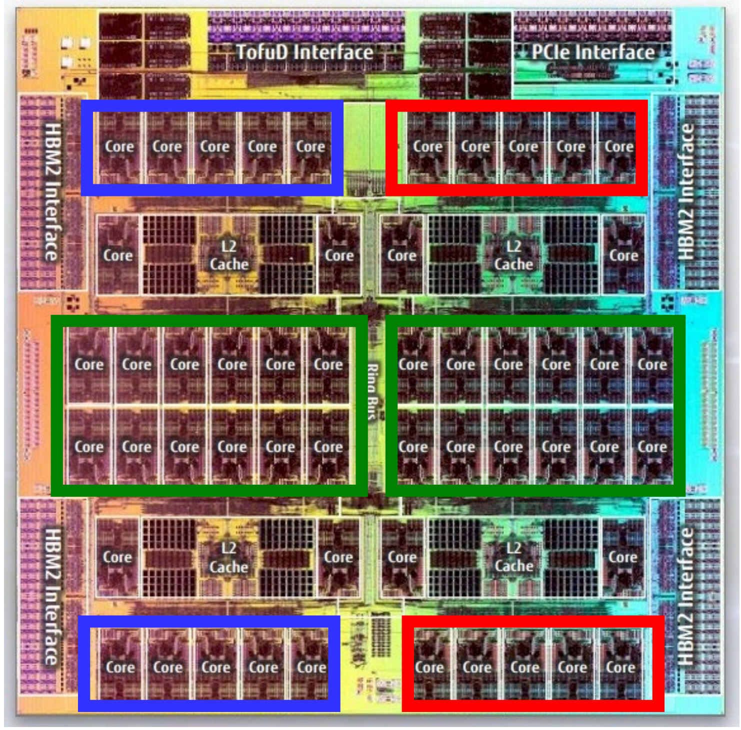
1. Ever increasing hardware parallelism

   - many-core processors, compute clusters, data centers
   - not all hardware resources will go into performance (some *should* go into assurance)

2. Ever increasing software complexity

   - *machine-learned components*, cloud connectivity
   - the static "verification gap" will only increase (software complexity grows faster than verification capability)

The Future Use of CPUs and GPUs ?!



instrument

compute

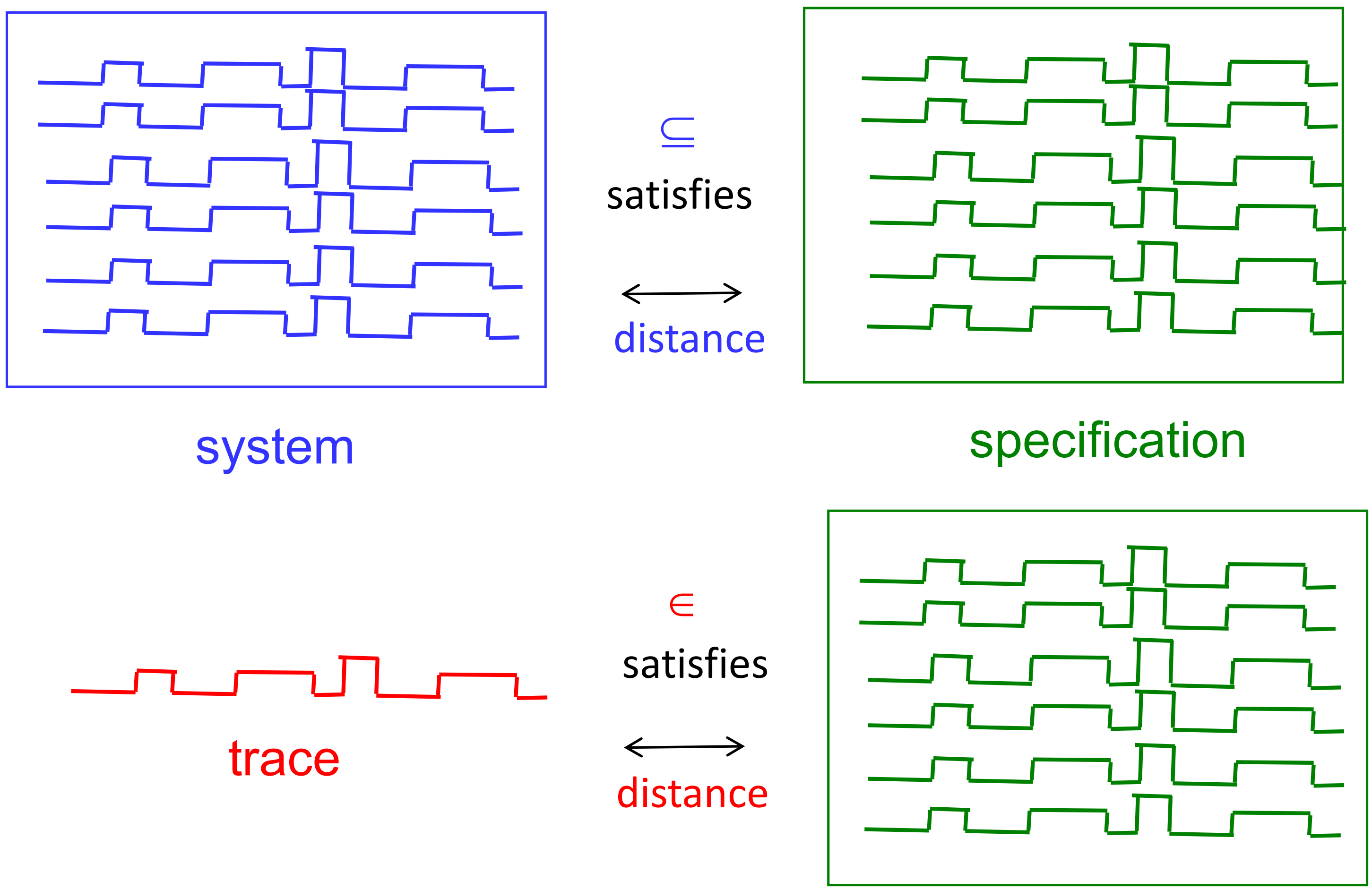
monitor

Two Observations about Runtime Verification

1. Static verification is about all possible infinite traces, runtime verification is about one *long* observed trace

- □ formalisms and methods should reflect this difference
  (the difference between emptiness and membership checking;
  over the *long* run, possibility becomes probability)
- □ opening for quantitative, statistical, and approximate methods
  (distance measures between traces rather than between systems)

Static versus Runtime Verification



Two Observations about Runtime Verification

1. Static verification is about all possible infinite traces, runtime verification is about one *long* observed trace

- □ formalisms and methods should reflect this difference
  (the difference between emptiness and membership checking;
  over the *long* run, possibility becomes probability)
- □ opening for quantitative, statistical, and approximate methods
  (distance measures between traces rather than between systems)

2. To increase public acceptance and trust of monitors, they must be third-party + unintrusive + online

- □ third-party means that system and monitor are designed *independently*
- □ unintrusive means *black-box* and *low-overhead*
- □ online means *real-time* and *best-effort*

Brief History of Boolean Monitoring

Observations $\Sigma = \{a, b, t, o\}$

Response $\square(a \rightarrow \lozenge b)$

Trace $o\ t\ t\ o\ a\ t\ o\ b\ o\ o\ t\ o\ a\ o\ o\ a\ o\ t\ o\ t\ o\ o\ o\ b\ o\ t\ o$ ...

Boolean verdict $\bot\ \bot\ \bot\ \ldots$

Response is live (every finite trace can be extended to satisfy response).
Response is co-live (every finite trace can be extended to violate response).

Response is not Pnueli-Zaks monitorable
(no finite trace allows a positive or negative verdict).

PZ monitorable $\supseteq$ Boolean combinations of safety properties
[Bauer-Leucker-Schallhart, Falcone-Frnandez-Mounier, Diekert-Leucker].

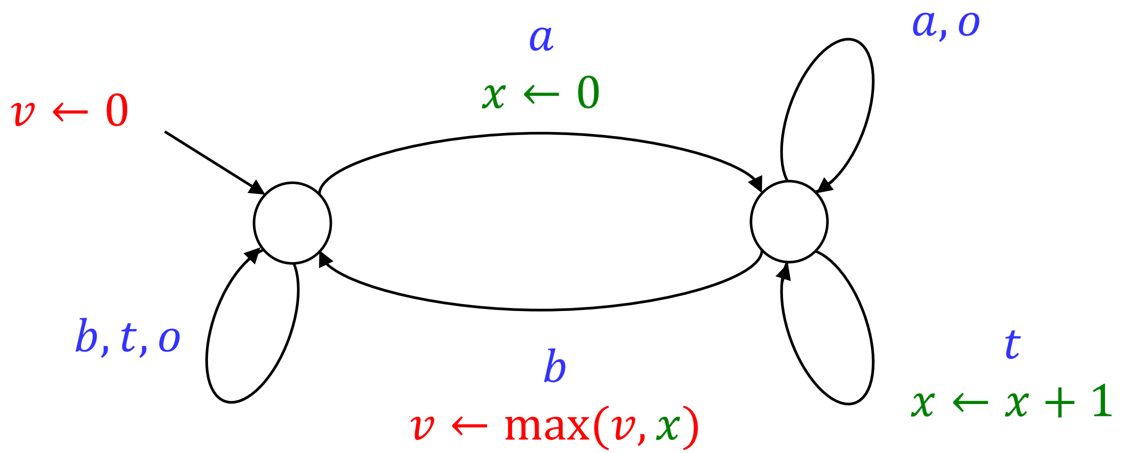
Quantitative Monitoring

Observations $\Sigma = \{a, b, t, o\}$

Max response time $min_t \, \Box(a \to \Diamond_{\leq t} \, b)$

Trace $o\ t\ t\ o\ a\ t\ o\ b\ o\ o\ t\ o\ a\ o\ o\ a\ o\ t\ o\ t\ o\ o\ o\ b\ o\ t\ o$ ...

Quantitative verdict 0 1 2



counter

Quantitative Monitoring

Observations $\Sigma = \{a, b, t, o\}$

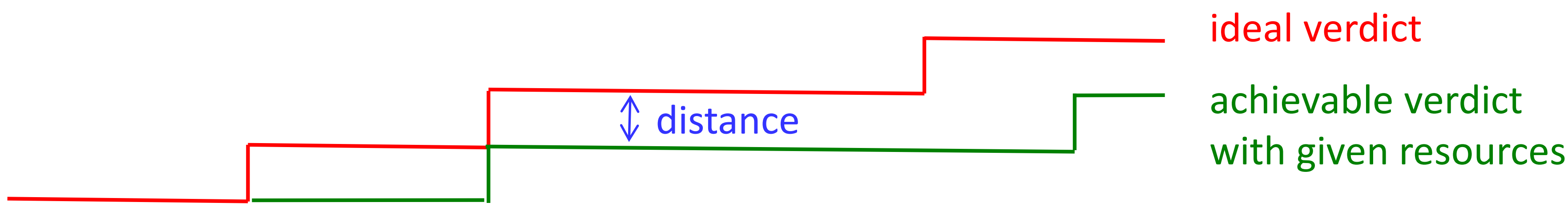
Max response time $min_t \, \square(a \rightarrow \Diamond_{\leq t} \, b)$

Trace $o\ t\ t\ o\ a\ t\ o\ b\ o\ o\ t\ o\ a\ o\ o\ a\ o\ t\ o\ t\ o\ o\ o\ b\ o\ t\ o \ldots$

Quantitative verdict 0 1 2

Approximate verdict 0 1 1

Delayed verdict 0 1 2



Quantitative Monitoring [LICS'21]

Observation alphabet	Σ
Value cpo	Λ
Quantitative property	$p: \Sigma^\omega \to \Lambda$
Verdict function	$v: \Sigma^* \to \Lambda$
Infinite trace	$w \in \Sigma^\omega$

smallest value such that every larger value is seen only finitely often ("least eventual upper bound")

Quantitative monitoring can be about limits:

verdict v monitors property p on trace w from below if $limsup_i \{v(w_{0..i})\} = p(w)$.

Quantitative monitoring can be universal:

property p is monitorable from below if $\exists v : \forall w : v$ monitors w from below.

Quantitative monitoring can be approximate:

verdict u approximates verdict v with error ε if $\forall w : \forall i : |v(w_{0..i}) - u(w_{0..i})| \leq \varepsilon$.

Quantitative Monitoring

Compare monitors with regard to

1. precision
2. resource use

◇

Expressiveness of Counter Monitors

Observations $\Sigma_k = \{0, \ldots, k\}$

Traces Σ_k^ω

3 3 2 3 3 2 2 1 3 2 1 0 1 2 3 3 2 1 0 0 3 3 …

Property $P_k = \{w \in \Sigma_k^\omega : \forall \pi \prec w : \forall i < k : \#(i+1, \pi) \geq \#(i, \pi)\}$

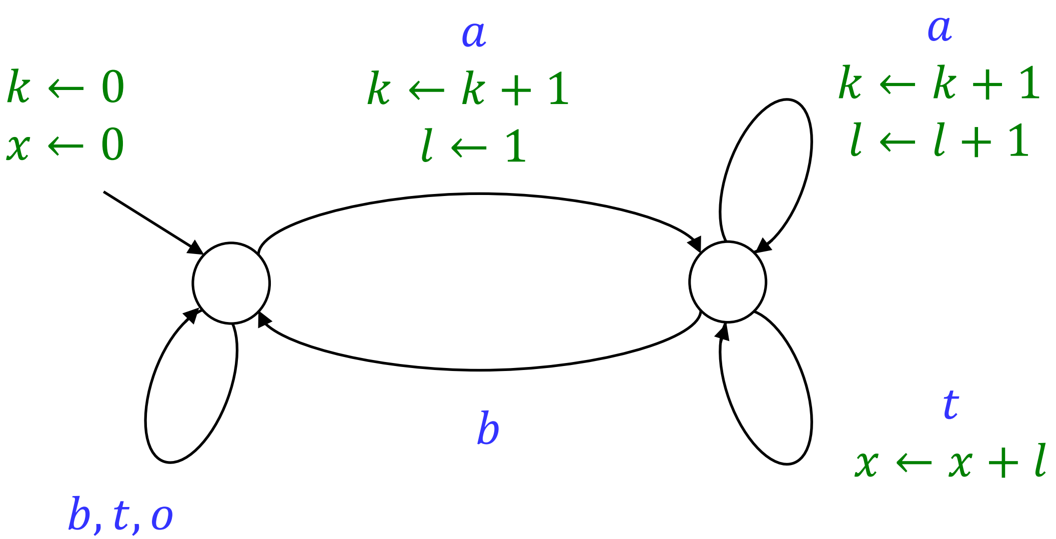
"Every prefix contains as many 1s as 0s, as many 2s as 1s, as many 3s as 2s, etc."

Theorem [LICS'18]:

For all k, the property P_k can be real-time monitored with $k+1$ counters but not with k counters.

(In contrast, in computability every Turing machine can be simulated by two counters.)

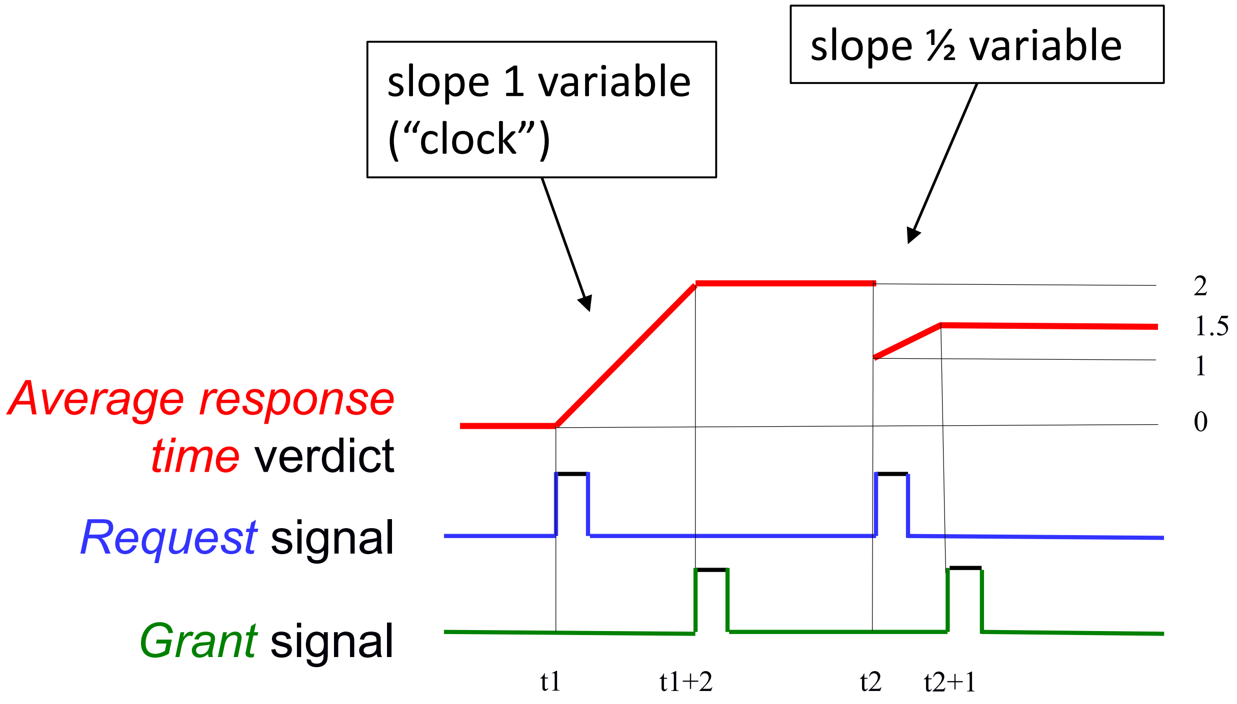
Monitoring Average Response Time



$$v(x, k) = x/k$$

Requires addition and division.
Led to original results in static verification [SAS'16].

Quantitative Monitoring in Continuous Time



Quantitative Monitoring

Compare monitors with regard to

1. precision
2. resource use
3. strength of assumptions

An assumption $A \subseteq \Sigma^{\omega}$ restricts the universe of possible traces [RV'20].
How do assumptions arise?

1. Knowledge about the system (predictive monitoring)
2. Knowledge about the environment (e.g. $\Box \Diamond t$)
3. Information from other monitors (decentralized monitoring)

Limit Monitoring [CSL'20]

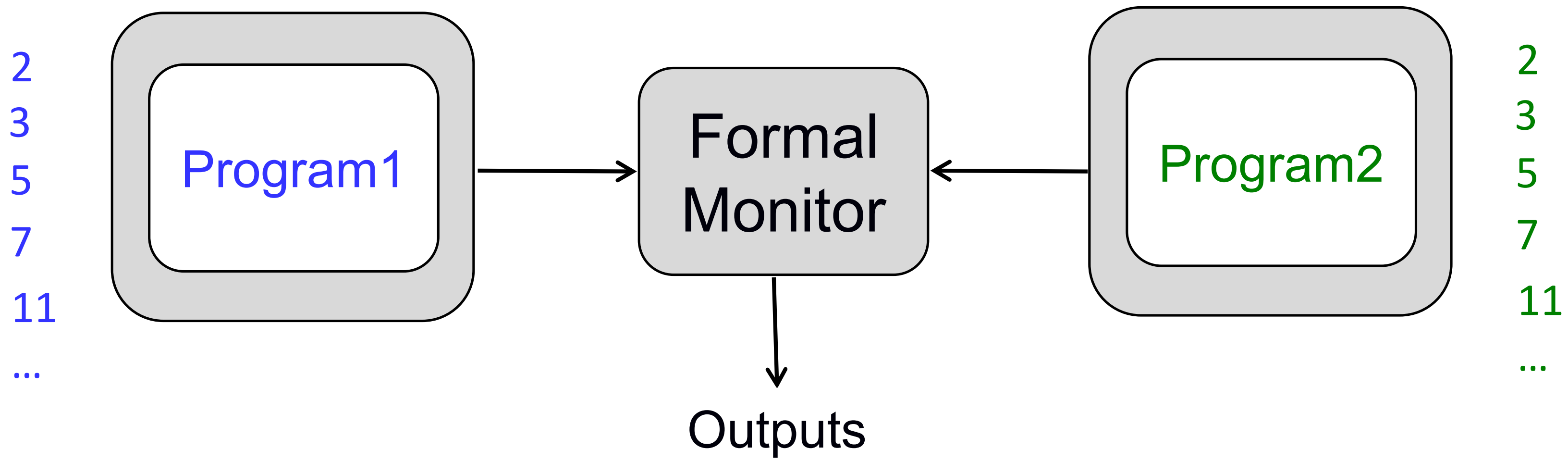
- mode $m(\pi)$ is the most common letter in $\pi \in \Sigma^*$
- for $w \in \Sigma^\omega$ generated by finite connected Markov chain (the "assumption"), $m(w) = \lim_i m(w_{0..i})$ converges with probability 1

Real-time monitoring the mode: requires |Σ| counters

Limit monitoring the mode: can be done by 4 counters

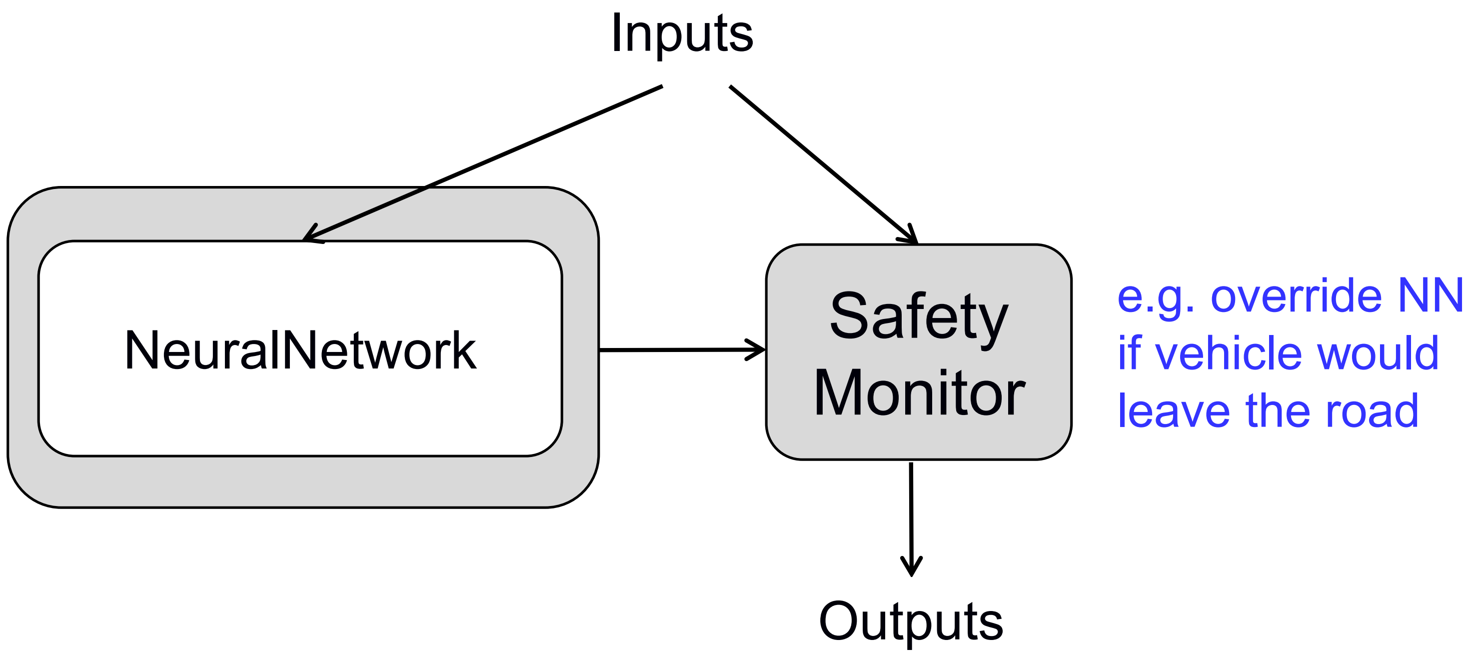
- partition $w \in \Sigma^\omega$ into segments $\alpha_0\alpha_1\alpha_2\alpha_3$ ... of increasing length
- while reading w, with each α_i maintain a current "hypothesis" $m_i \in \Sigma$ for the mode
- cycle round-robin through the letters in Σ, looking at one letter σ_i per segment α_i
- if σ_i occurs more often than m_i in α_i, then $m_{i+1} = \sigma_i$ else $m_{i+1} = m_i$

Differential Monitoring [RV'21]



- □ assurance through redundancy ("overengineering")
- □ programs as specifications

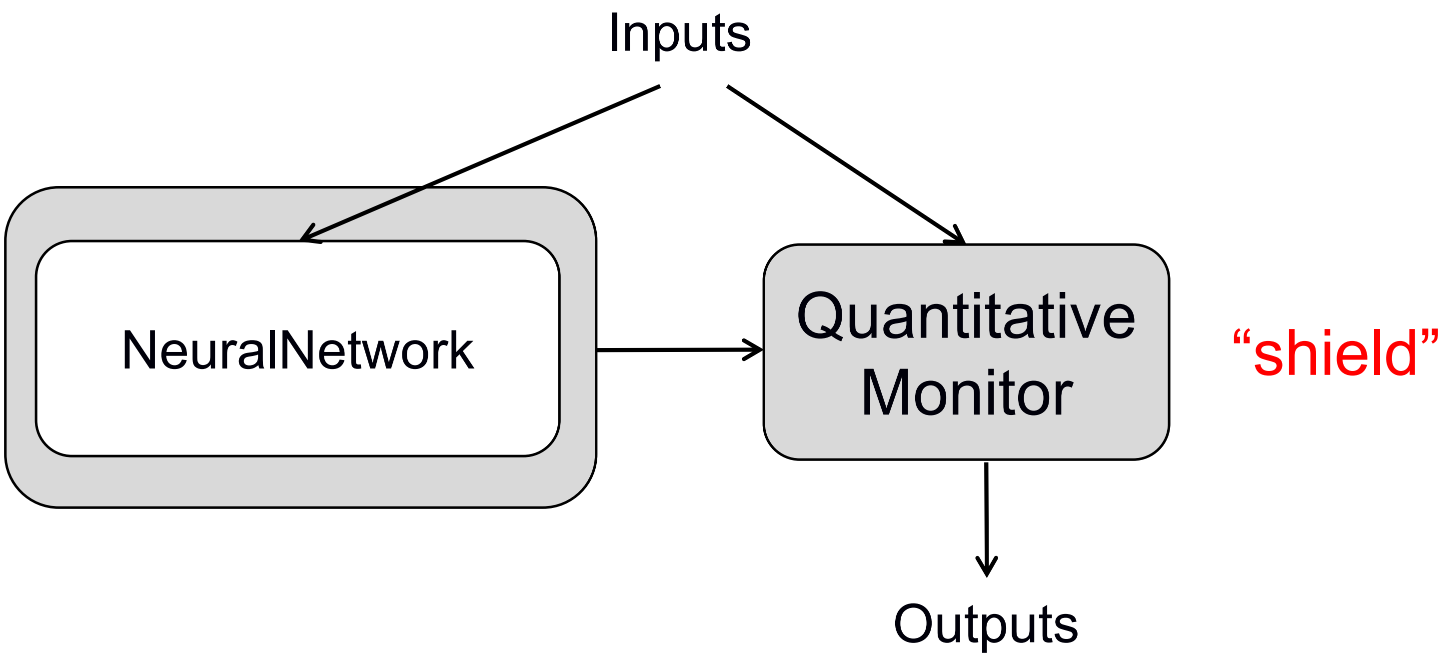
Autonomous Vehicles: Monitors as Shields



- □ neural network computes control actions
- □ monitor overrides actions that violate safety

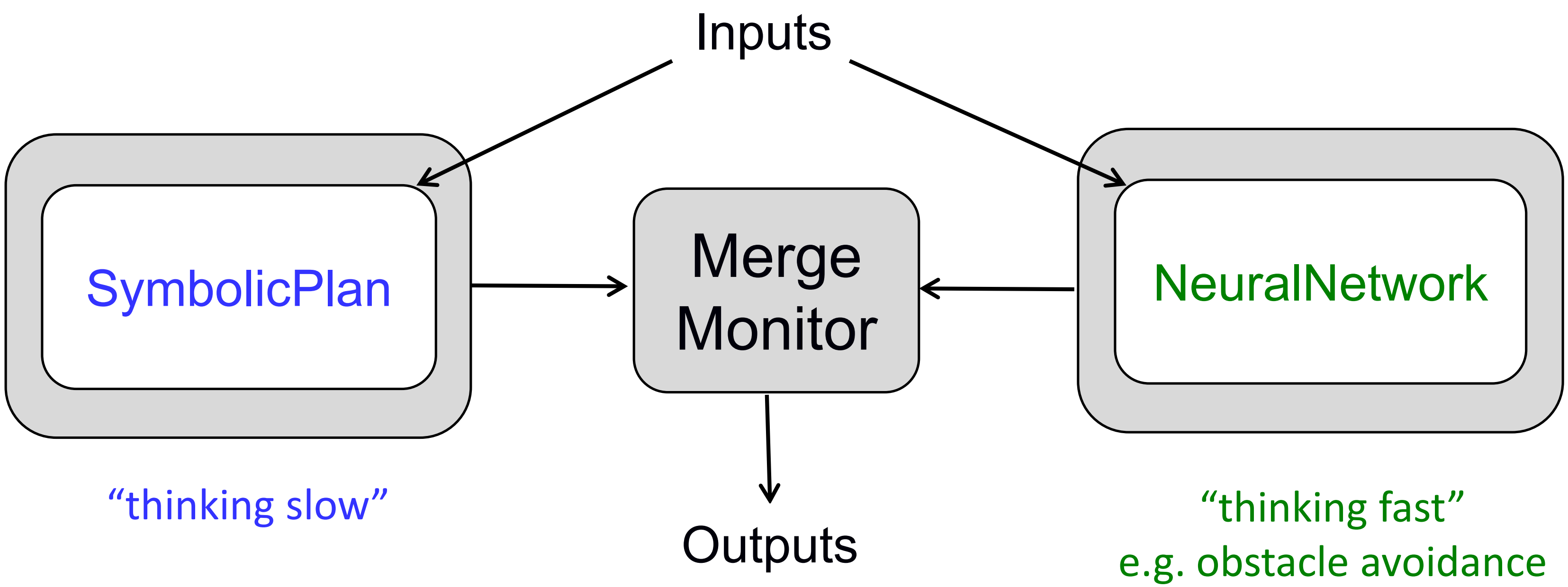
[Bloem-Koenighofer-Koenighofer-Wang]

Beyond Safety: Quantitative Shields [CAV'19]



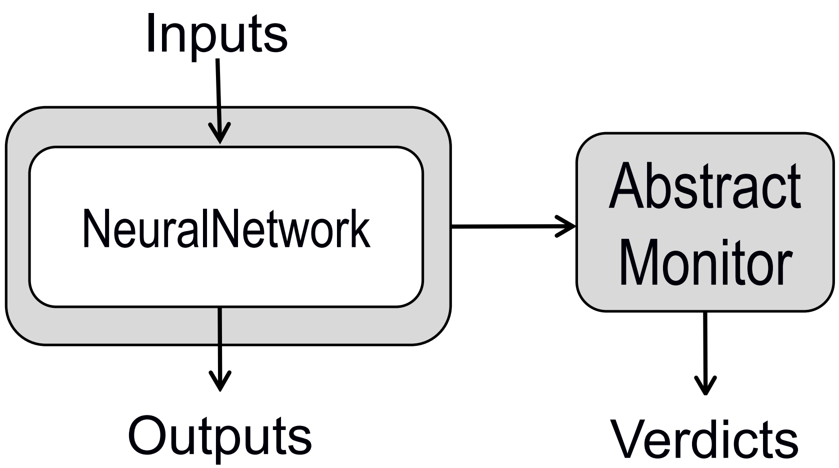
- quantitative rewards and interference penalties
- shield specification using weighted automaton
- shield synthesis by solving stochastic games (controller+plant against shield)

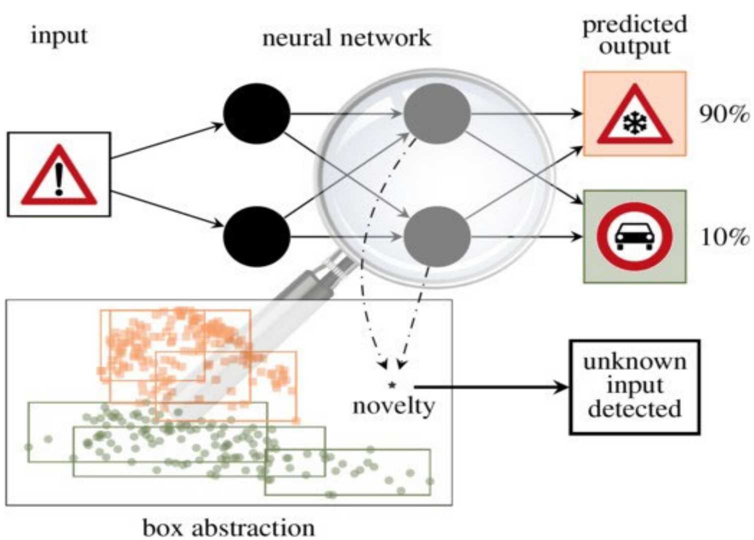
Planning: Monitors as Arbiters



- neural network computes short-term actions
- symbolic controller follows long-term plan
- monitor arbitrates both decisions

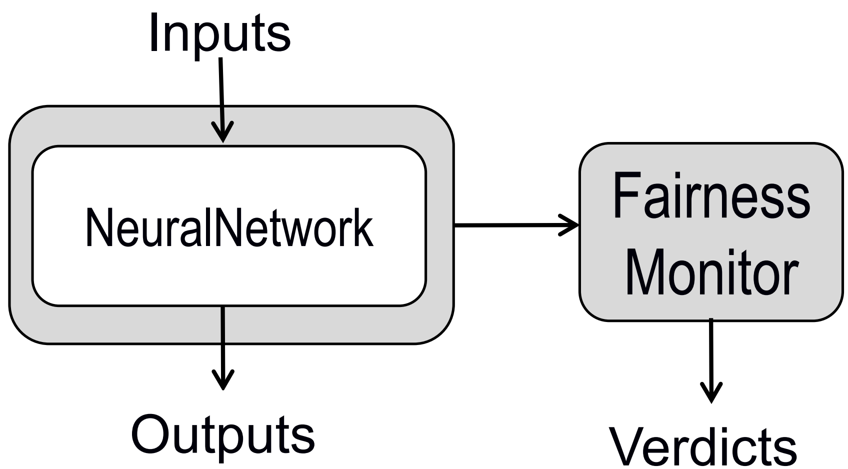
Classification: Monitors as Novelty Detectors [RV'21]





- □ monitor watches if predictions are similar to previously observed patterns
- □ when novelties are detected, network may need retraining

Algorithmic Fairness: Monitors as Watchdogs



Demographic parity

$$\frac{\Pr(\ output = 1 \mid protectedInputAttribute = 0\ )}{\Pr(\ output = 1 \mid protectedInputAttribute = 1\ )} \cong 1$$

can be monitored using frequency counters.

Summary

Static formal methods have scalability issues for neural networks.

A promising approach is the LEARNER + VERIFIER architecture,
which uses machine learning for hypothesis generation of correctness certificates.

Runtime methods can monitor not only safety violations,
but also *quality, limits*, and *fairness.*
They can enforce not only safety (shielding),
but also *progress* (planning).

References

[NeuIPS'21] Infinite time horizon safety of Bayesian neural networks
[AAAI'22] Stability verification in stochastic control systems via neural network supermartingales

[TACAS'20] How many bits does it take to quantize your neural network?
[AAAI'21] Scalable verification of quantized neural networks

[CAV'19] Runtime optimization for learned controllers
[ECAI'20] Outside the box: Abstraction-based monitoring of neural networks
[RV'21] Into the unknown: Active monitoring of neural networks