Compositional Verification and Runtime Monitoring for Learning-Enabled Autonomous Systems

Corina S. Pasareanu, NASA Ames, KBR, CMU CyLab

Thanks!

Ravi Mangal, CMU CyLab
Divya Gopinath, NASA Ames, KBR
Sinem Getir Yaman, York University, UK
Calum Imrie, York University, UK
Radu Calinescu, York University, UK
Huafeng Yu, Boeing, USA

Problem

- Autonomous systems increasingly use Deep Neural Networks (DNNs) for perception
  - Need to be highly reliable
- Reasoning about “closed loop” autonomous systems is very difficult
  - High complexity of the DNN (thousands or millions of parameters)
  - Complexity of the high-definition cameras
  - Complexity of the environment, subject to random perturbations


Our Approach

Key idea:

- **Abstract** away the hard-to analyze components:
  - Perception DNN, camera, environmental dynamics
- Replace them with **probabilistic or worst-case abstractions**
- Model other components (controller, plant) using conventional techniques
- **System becomes amenable to formal verification with off-the-shelf tools**
- Approach is **compositional**
  - **Conventional components analyzed separately from perception components**

This talk:

- **Probabilistic (average-case) analysis**: provides probabilistic guarantees
- **Worst-case analysis**: provides strong (non-probabilistic) guarantees

Case Study: TaxiNet

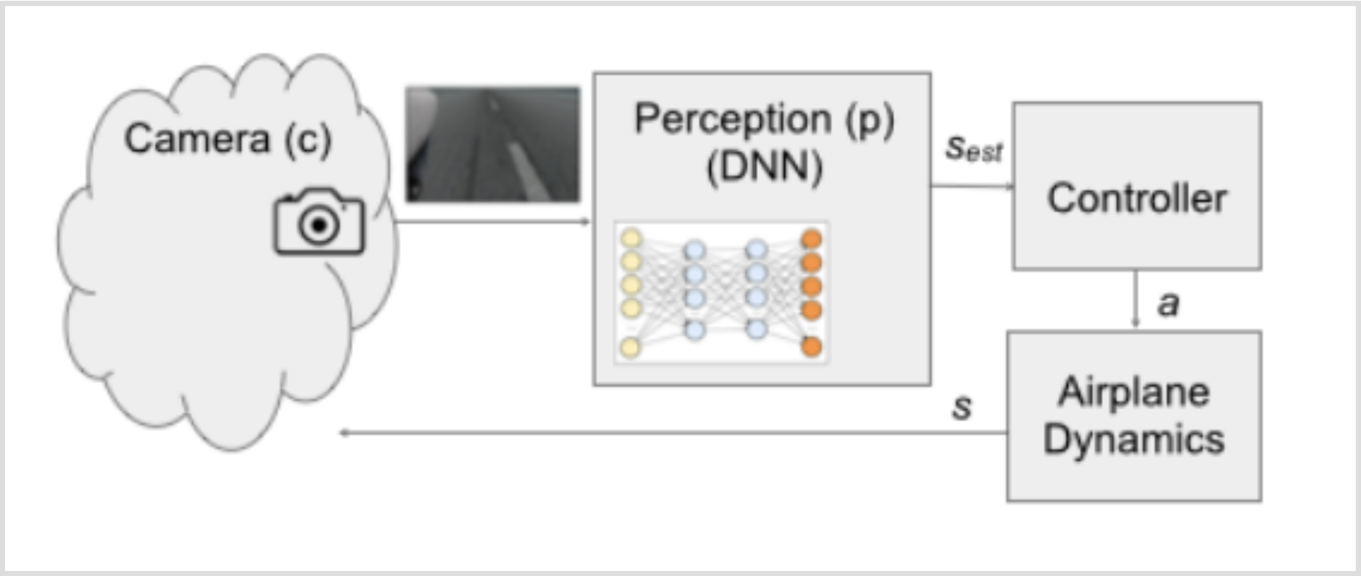
- Neural network designed to take a picture of the runway as input and return the plane's position w.r.t. the middle of the runway
- Returns two numerical outputs
  - Cross-track error (`cte`): The distance of the plane from the middle line
  - Heading error (`he`): The angle of the plane w.r.t. the runway

- Simple scenario:
  - From an initial state, keep straight line for a finite number of steps
- **Properties**:

  (Property 1) Airplane does not go off runway: **|`cte`| ≤ 8 meters**

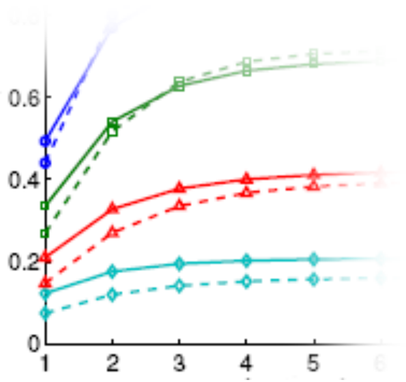
  (Property 2) Airplane does not turn more than certain degree: **|`he`| ≤ 35 degrees**

Autonomous Line Tracking System



- State ***s***: actual values of **(cte,he)**
- Estimated state $\boldsymbol{s_{est}}$: estimated values of **(cte,he)** as returned by the DNN

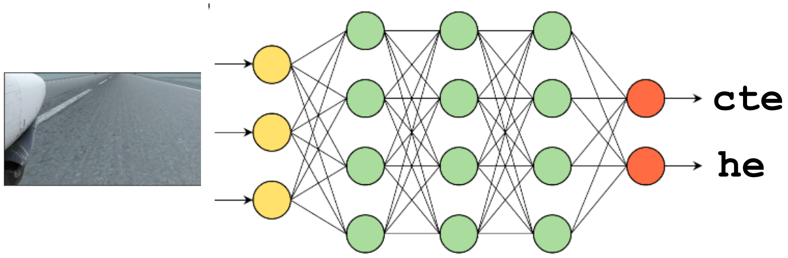
Discrete Models



- We build a discrete-state model of the system for analysis
  - Discrete controller
  - Discrete model of airplane dynamics
- **System state**: Real-valued (**`cte, he`**)
- **Discretize** system state (both actual and estimated) as dictated by controller logic
- The regression outputs of TaxiNet are discretized to view the model as a ***classifier*** which predicts the plane's position in discrete states

Discretized View of TaxiNet

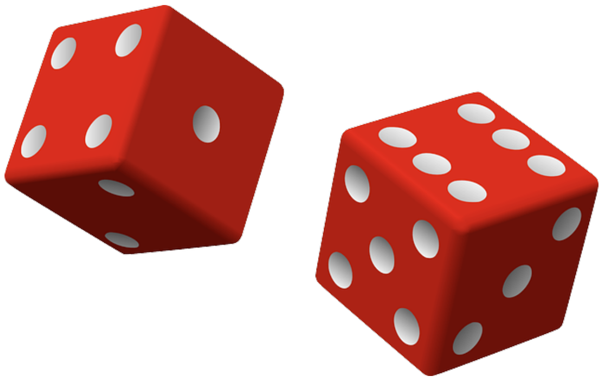
- Taxinet DNN model
  - Input images: RGB color images, 360 × 200 pixels
  - 24 layers CNN, 3 dense layers before output
  - Representative dataset with 11108 images
  - Mean Absolute Error (MAE): cte : 1.185, he: 7.86
- Discretization of outputs to view the model as a classifier
- Values outside the intervals: error states (encoded as “-1”)



$$\underline{\texttt{cte}} = \begin{cases} 3 \text{ if } -8.0 \text{ m} <= \texttt{cte} < -4.8 \text{ m} \\ 1 \text{ if } -4.8 \text{ m} <= \texttt{cte} < -1.6 \text{ m} \\ 0 \text{ if } -1.6 \text{ m} <= \texttt{cte} <= 1.6 \text{ m} \\ 2 \text{ if } 1.6 \text{ m} < \texttt{cte} <= 4.8 \text{ m} \\ 4 \text{ if } 4.8 \text{ m} < \texttt{cte} <= 8.0 \text{ m} \end{cases}$$

$$\underline{\texttt{he}} = \begin{cases} 1 \text{ if } -35.0^\circ <= \texttt{he} < -11.67^\circ \\ 0 \text{ if } -11.67^\circ <= \texttt{he} <= 11.66^\circ \\ 2 \text{ if } 11.66^\circ < \texttt{he} <= 35.0^\circ \end{cases}$$

Probabilistic Analysis [CAV'23]



Probabilistic Abstraction for Perception

Probabilistic abstraction maps **actual** system states to (a distribution over) **predicted** states

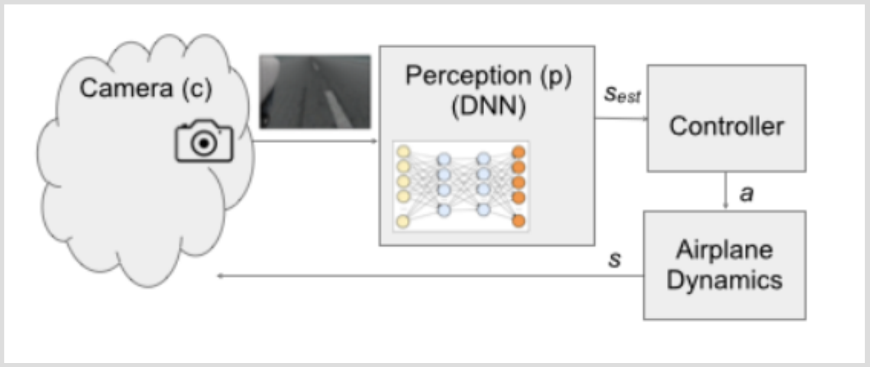
- Abstraction **linear** in the size of the output of the DNN, independent of the number of DNN parameters, the camera or the environment

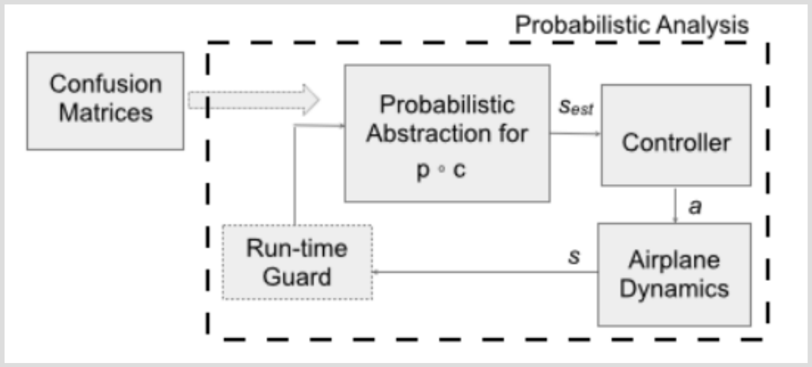
Why probabilistic view?

- Camera maps one 3D vehicle position to a distribution of images
- Different environment conditions (light, contrast, skid marks etc)

We leverage DNN-specific analysis (e.g., robustness) to define run-time guards

- Refine the abstraction and increase the safety of the system





Probabilistic Abstraction for Perception

Probabilistic abstraction:

- Maps every (discrete) system state to every (discrete) estimated state
- Transition probabilities estimated based on confusion matrices for perception DNN, measured on **“representative”** data set

State: (**`cte,he`**)

		Predicted		
	Total = 11108	**0**	**1**	**2**
Actual	**0**	4748	2139	148
	1	91	2010	0
	2	744	211	1017

Table 1: Confusion Matrix for **he**

DTMC code:

```
[] he=0 → 0.675: (he_est'=0) + 0.304: (he_est'=1) + 0.021: (he_est'=2);
[] he=1 → 0.043: (he_est'=0) + 0.957: (he_est'=1) + 0.0: (he_est'=2);
[] he=2 → 0.377: (he_est'=0) + 0.107: (he_est'=1) + 0.516: (he_est'=2);
```

DNN Checks as Run-time Guards

- Many techniques for DNN analysis
  - Robustness, safety, confidence, out-of-distribution detection, Prophecy, etc.
  - Can be black box or white box; complete or incomplete
  - How can we leverage these off-the-shelf analysis techniques to improve the safety of the overall system?
- Our approach
  - Uses DNN Checks as run-time guards
  - For inputs that pass the checks, the DNN is more likely to be correct/accurate.
  - For TaxiNet, we use rules extracted with Prophecy
  - Out of 11108 inputs, 9125 inputs (82.1%) pass the DNN check:

```
i:[0..M] init 0;
[]  pc=0 & i<M → 0.821: (v'=1) & (pc'=1) & (i'=0) + 0.179: (v'=0) & (i'=i+1);
```

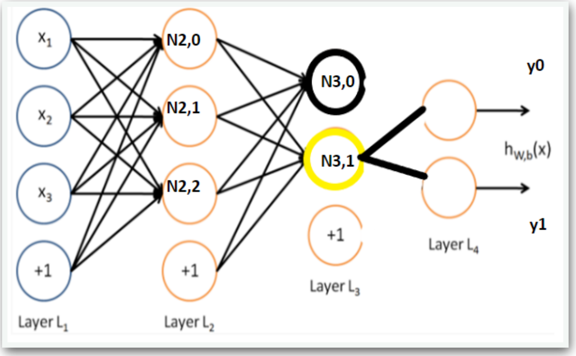
The abstract map for state variables he and cte is only computed for the inputs that pass the check (i.e., for v = 1) based on newly computed confusion matrices

Prophecy Rules as Run-time Guards

- Inferred automatically from DNN neuron activations [ASE'19]
  - Intent is to capture properties on the semantic features the network has learnt
  - Built with **decision-tree learning** over activations collected on training data, validated on test data
- Rule: Pre => Post
- Pre is a condition on neuron values at some layer; Post = "mis-prediction"

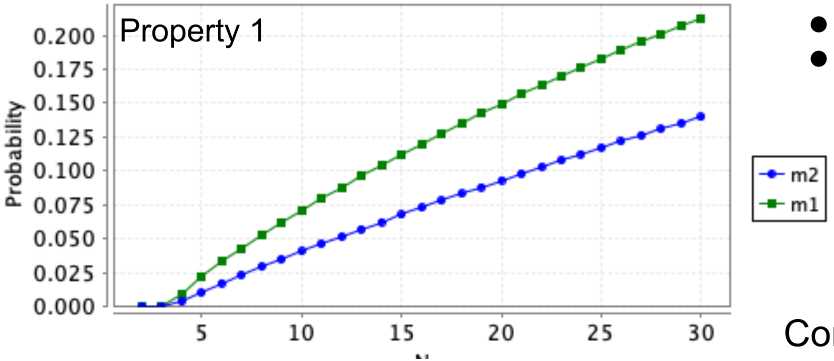
$$N_{1,85} <= -0.998 \ \wedge \ N_{2,50} <= 3.31 \ \wedge \ N_{1,84} <= -0.994 \ \wedge \ N_{1,15} > -0.999$$
$$\wedge \ N_{1,21} <= 1.711 \wedge N_{1,70} <= 11.088 \wedge N_{1,51} > -0.999 \wedge N_{1,21} > -0.637 \implies$$
$$|\mathtt{cte}^* - \mathtt{cte}| > 1.0 \ meters \ \vee \ |\mathtt{he}^* - \mathtt{he}| > 5 \ degrees$$

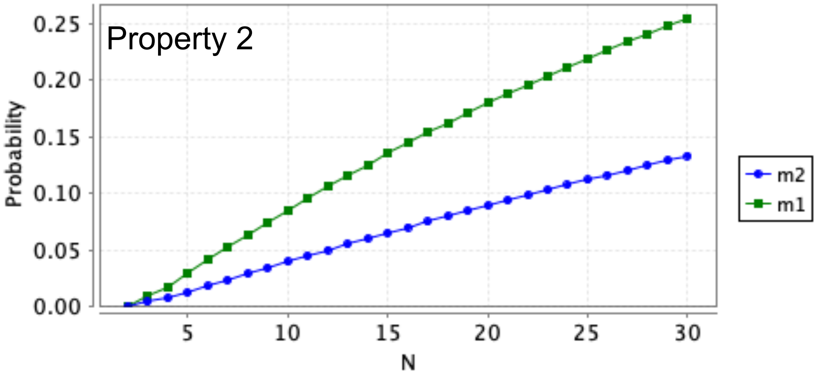
- If an input satisfies Pre it is considered to violate the runtime check
- Can be evaluated efficiently during forward pass of DNN
- If the check is violated M times, go to "abort" state
  - e.g., hand over control to the pilot



(N3,0 = 0 /\ N3,1 > 0) => y0 < y1 (label 1)

Experiments with PRISM





Analyzed two models:

- m1 (no run-time guard)
- m2 (with run-time guard)
  - Rules characterizing inputs where the model gives mis-predictions
  - A rule is of the form **Pre => Post**
  - Pre is a condition in the latent space; Post is a condition on the output
  - An input passes the guard if it is not “rejected” by the rule
  - Extracted using **Prophecy** from the dense layers of the model

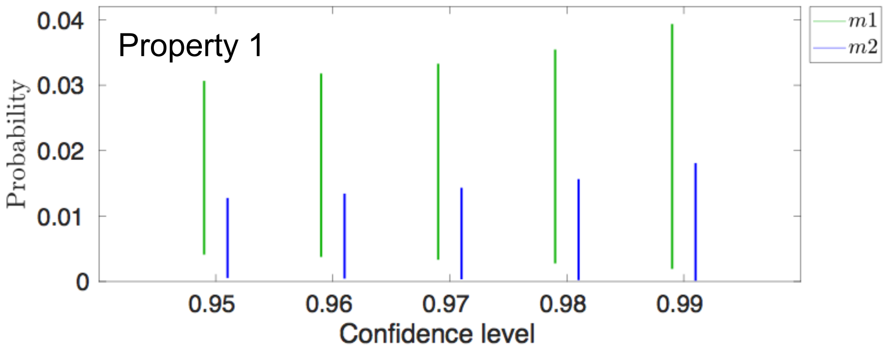
Controller and dynamics are the same for both models

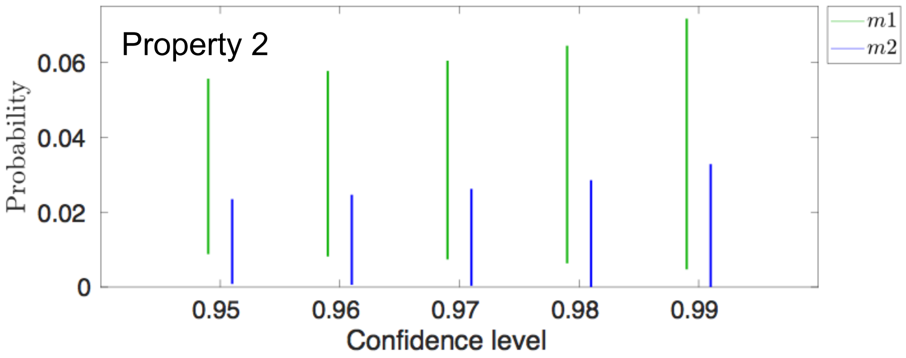
(Property 1) **P =?[F (`cte` = −1)]**

(Property 2) **P =?[F (`he` = −1)]**

Experiments with FACT: Confidence Interval Analysis

- Probabilistic abstractions based on empirical estimates of probabilities
  - Lack **statistical guarantees;** can be off from true probabilities
- We compute **confidence intervals**
  - For the transition probabilities
  - For the probability that the safety properties are satisfied
- FACT tool:
  - Synthesizes a $(1 - \delta)$-**confidence interval** $[a, b] \subseteq [0, 1]$ for the probability that a property φ is satisfied, given a set of observations (based on confusion matrices)



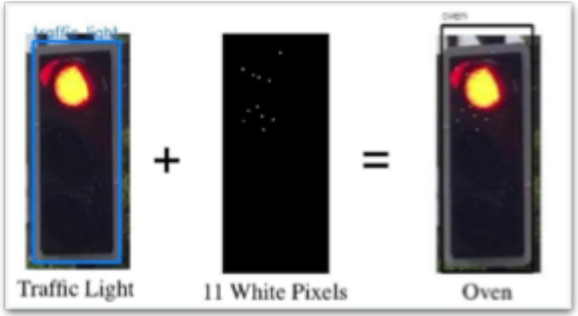


Summary

- Experiments demonstrate the feasibility of our approach
  - Analysis of DNN working side-by-side with conventional components (controller, dynamics)
  - Abstraction **separates** the concerns of DNN and conventional system development and evaluation
  - Analysis incorporates accuracy/confusion matrices results in the system-level analysis
- We provide **probabilistic guarantees**
  - Address gaps of quantitative evaluation for future AI certification
- Experiments show benefit of the run-time guards
- *Improved performance of the DNN translates into improved safety*

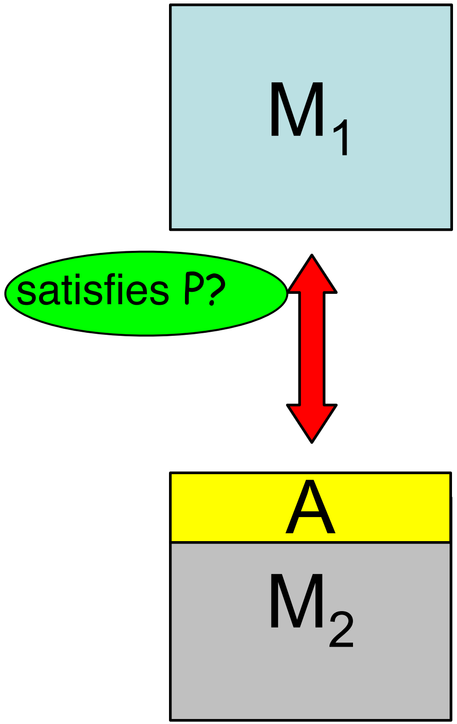
Discussion

- What about adversarial examples?
  - Use local robustness certifiers (such as CMU's Gloro) as run-time guards
- What about out-of-distribution inputs?
  - Use out-of-distribution detectors as run-time guards
- What about other rare events?
  - "Smarter" sampling, e.g. stratified sampling
- What if the data set is not "representative"?
  - **"Average-case" analysis**; the system should be safe at least in this average case!
  - Parametric probabilistic analysis: instead of using probabilities empirically derived from confusion matrices, generate them automatically from the analysis of the closed-loop system with **parametric** model for perception



Compositional Worst-Case Analysis

Compositional Verification



does system made up of M_1 and M_2 satisfy property P?

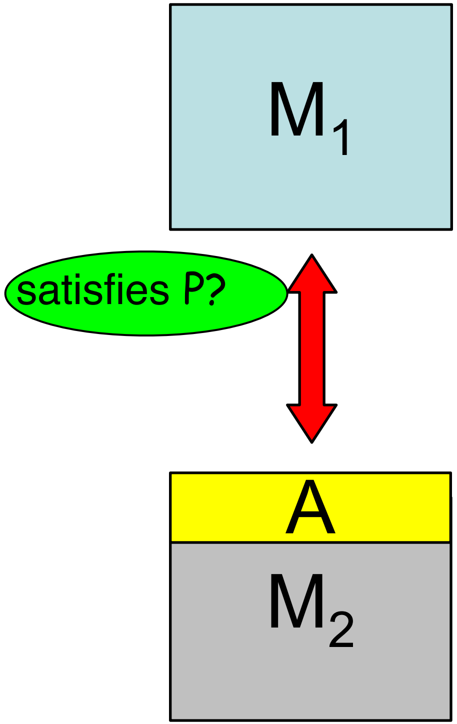
- check P on entire system: too many states!
- use the natural decomposition of the system into its components to break-up the verification task
- check components in isolation
- does M_1 satisfy P?

typically a component is designed to satisfy its requirements in specific contexts / environments

- assume-guarantee reasoning [Jones 83, Pnueli 85]

introduces assumption A representing M_1's “context” at the level of its interactions with the component

Assume-Guarantee Reasoning



reason about triples:

$$\langle A\rangle\ M\ \langle P\rangle$$

the formula is *true* if whenever M is part of a system that satisfies A, then the system must also guarantee P

simplest assume-guarantee rule

$$\frac{\begin{array}{c}\langle A\rangle\ M_1\ \langle P\rangle \\ \langle \mathit{true}\rangle\ M_2\ \langle A\rangle\end{array}}{\langle \mathit{true}\rangle\ M_1 \parallel M_2\ \langle P\rangle}$$

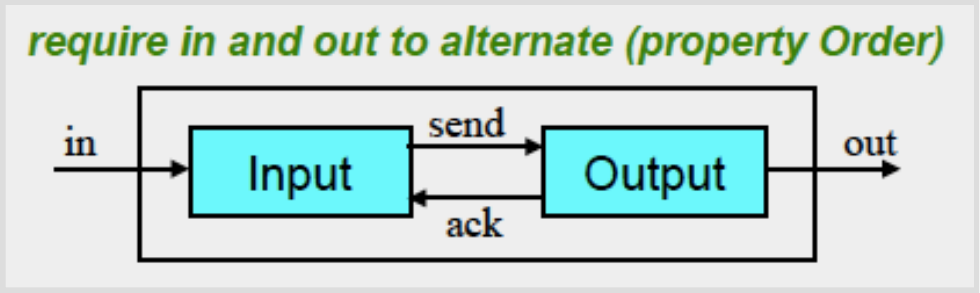
"discharge" the assumption

we synthesize the assumption automatically [ASE'02,TACAS'03]

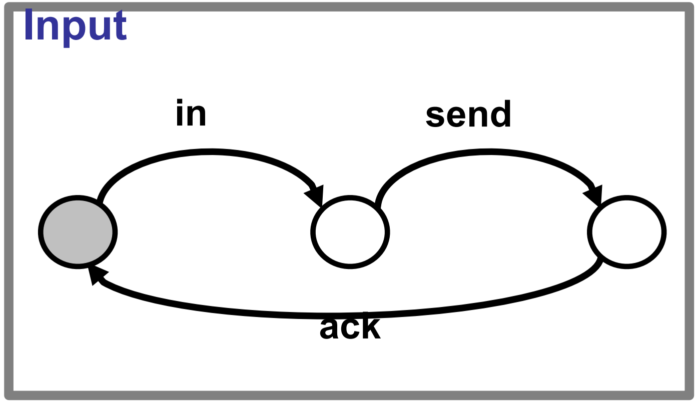
Formalisms

- components modeled as finite state machines (FSM)
  - FSMs assembled with parallel composition operator "||"
  - Synchronizes shared actions, interleaves remaining actions
- a safety property P is a FSM
  - P describes all legal behaviors
  - P_{err} – complement of P
    - make deterministic & complete P with an "error" state;
    - bad behaviors lead to error
  - component M satisfies P iff error state unreachable in $(M \mid\mid P_{err})$
- assume-guarantee reasoning
  - assumptions and guarantees are FSMs
  - $\langle A \rangle\ M\ \langle P \rangle$ holds iff error state unreachable in $(A \mid\mid M \mid\mid P_{err})$

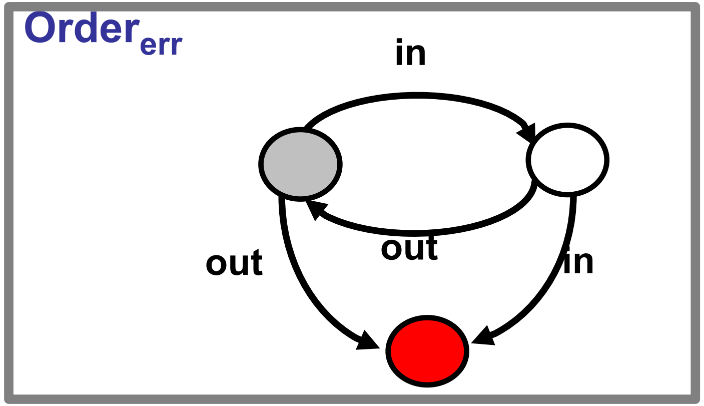
Example



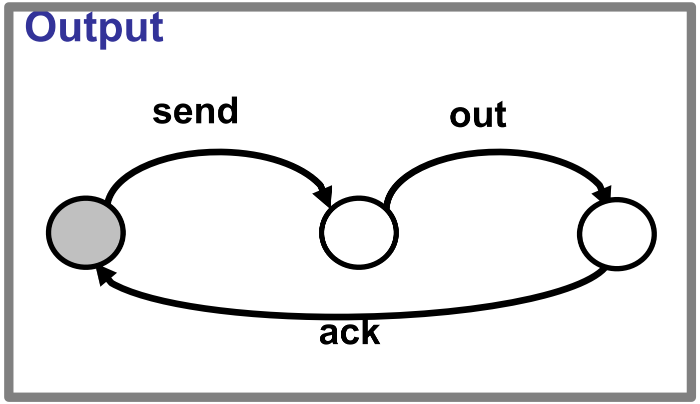
require in and out to alternate (property Order)
in
send
Input
Output
out
ack



Input
in
send
ack

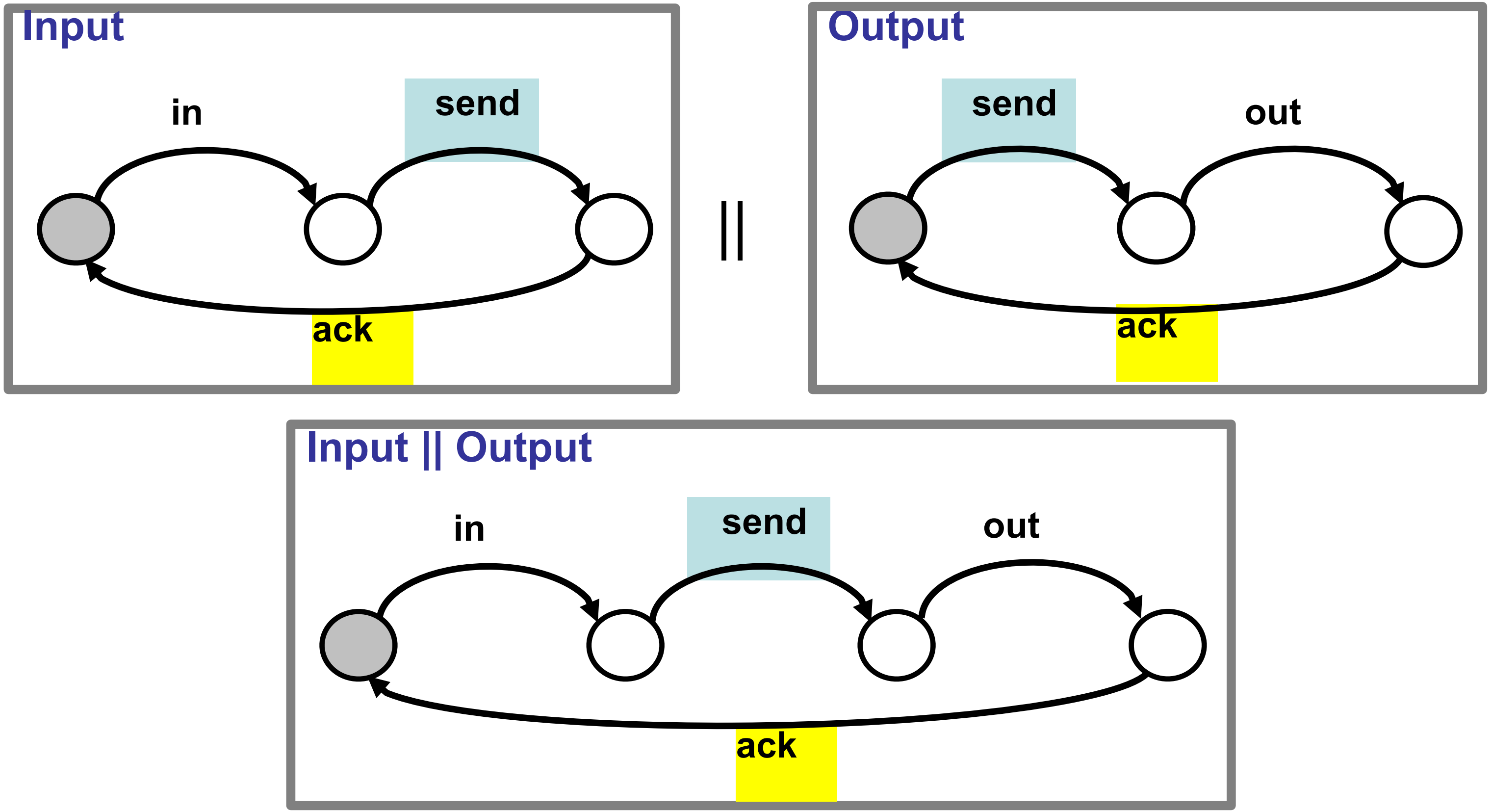


Order$_{err}$
in
out
out
in

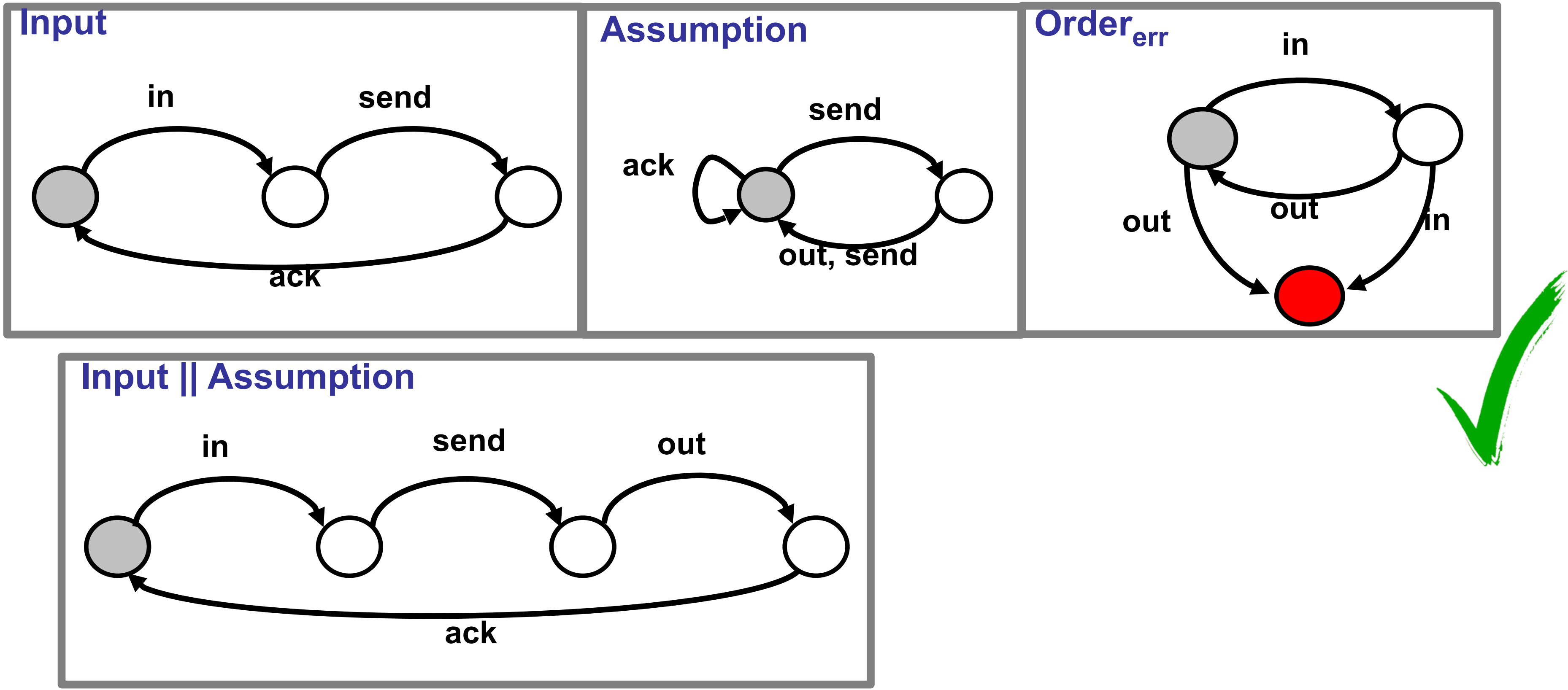


Output
send
out
ack

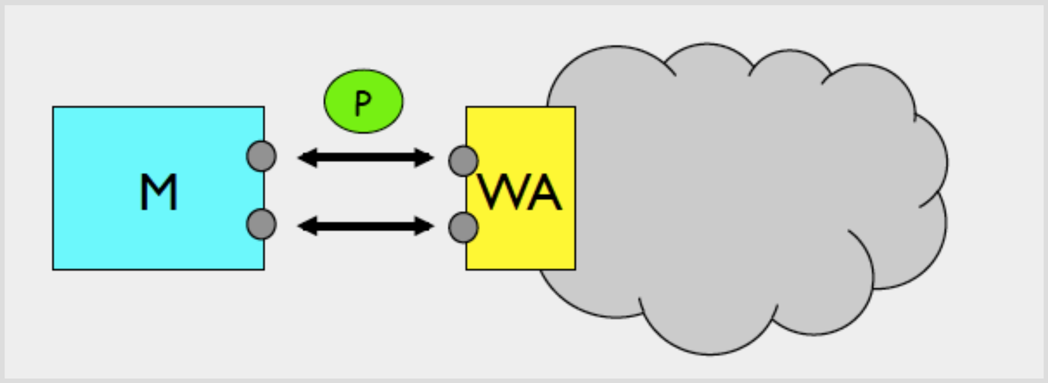
Parallel Composition



Assume-Guarantee Reasoning



Weakest Assumption [ASE'02]



- Inputs: Component M, property P, interface (alphabet) of M || P_{err} with its context
- Output: Weakest environment assumption WA such that ⟨WA⟩ M ⟨P⟩ holds
- Weakest assumption:
  - prevents component to go to error (safe)
  - is as permissive as possible
  - uses only interface actions

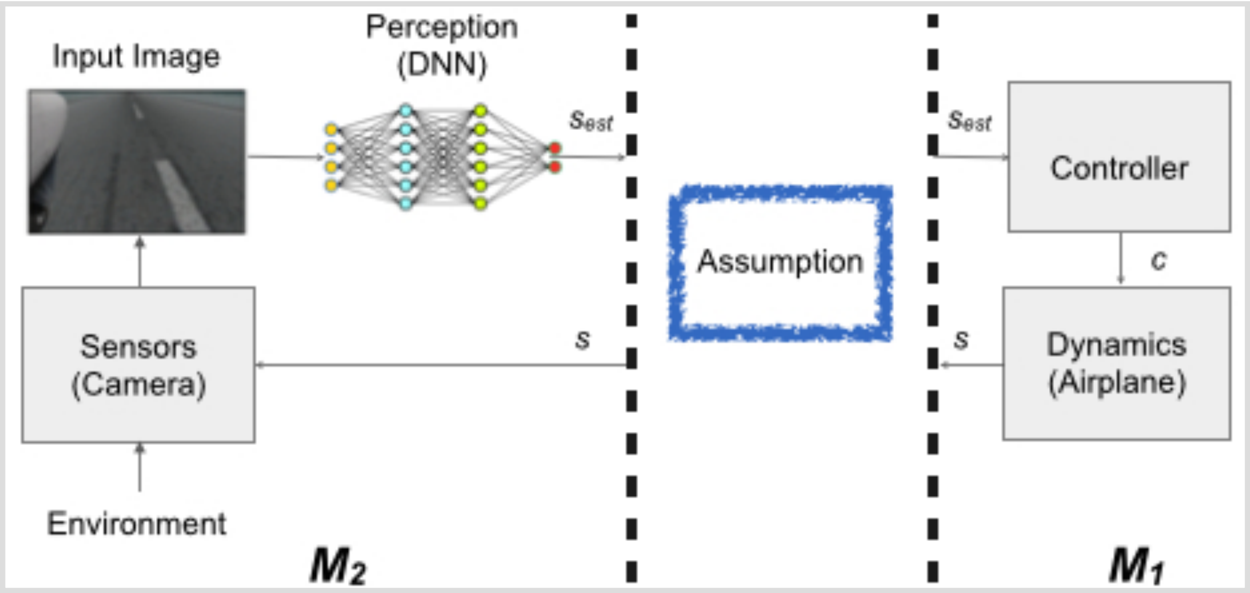
Giannakopoulou, D., Pasareanu, C.S., Barringer, H.: Assumption generation for software component verification. [ASE'02]

Weakest Assumption and Assume-Guarantee Reasoning

- Weakest assumption for M and P
  - for all environment components N: $\langle true \rangle\ M \| N\ \langle P \rangle$ iff $\langle true \rangle\ N\ \langle WA \rangle$
- Let's use WA (for M_1 and P) in the rule
  - if both $\langle WA \rangle\ M_1\ \langle P \rangle$ and $\langle true \rangle\ M_2\ \langle WA \rangle$ hold then $\langle true \rangle\ M_1 \| M_2\ \langle P \rangle$ holds
  - if $\langle true \rangle\ M_2\ \langle WA \rangle$ does not hold then $\langle true \rangle\ M_1 \| M_2\ \langle P \rangle$ does not hold

$$\frac{\begin{array}{c}\langle A \rangle\ M_1\ \langle P \rangle \\ \langle \mathit{true} \rangle\ M_2\ \langle A \rangle\end{array}}{\langle \mathit{true} \rangle\ M_1 \parallel M_2\ \langle P \rangle}$$

Assume-Guarantee Reasoning for the TaxiNet System



Property: safe operation. |`cte`| ≤ 8 meters and |`he`| ≤ 35 degrees

Compositional Analysis

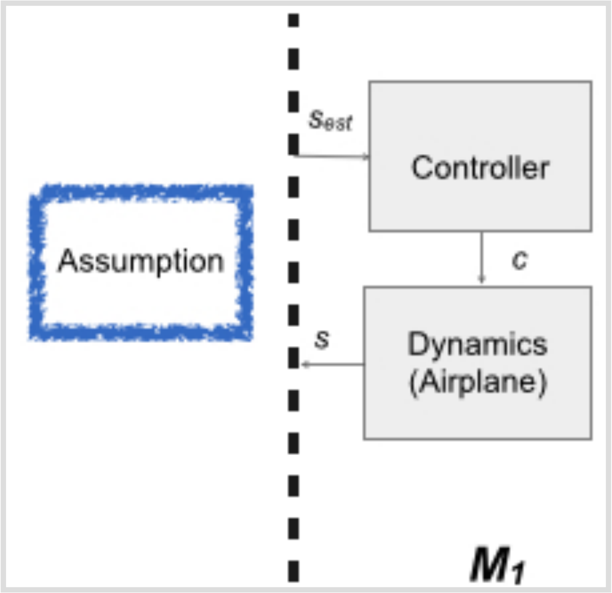
Optimistic View:

- Analyze the system assuming **ideal perception**
- Fix all errors due to controller and dynamics logic

Pessimistic View:

- Analyze the system in the **absence** of the DNN
- Implicit **worst-case** behavior: estimates can be arbitrarily wrong
- Accounts for **all possible** perturbations in the environment, distribution shifts, etc.
- Compute weakest assumption: [ASE'02] with small modifications

Assumption encodes all the DNN behaviors that **guarantee** that the autonomous system satisfies the property!



Property

How to “discharge” the assumption?

Formal verification is difficult (impossible?)

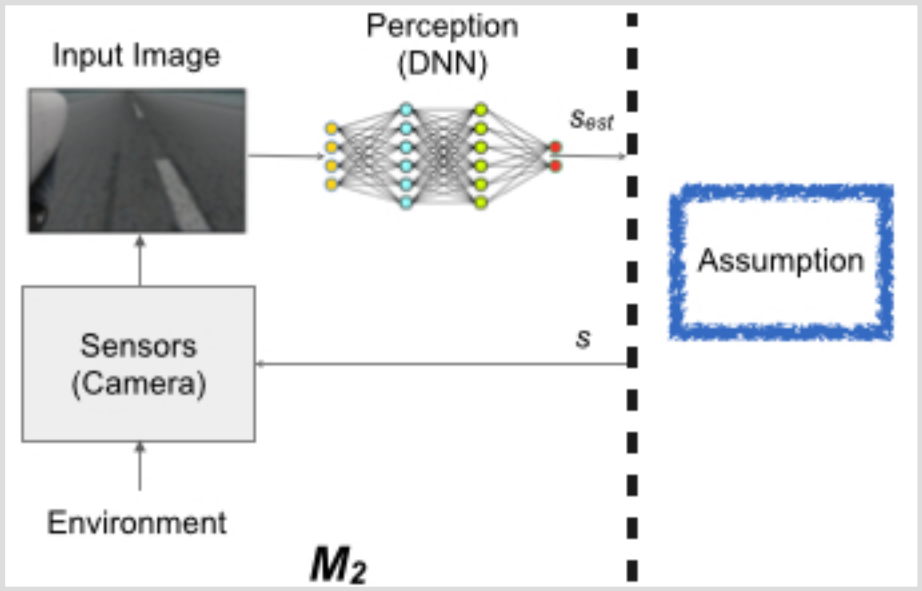
- DNN size (millions/billions of parameters)
- Modeling of all the possible environment conditions

Solution: run-time monitoring!

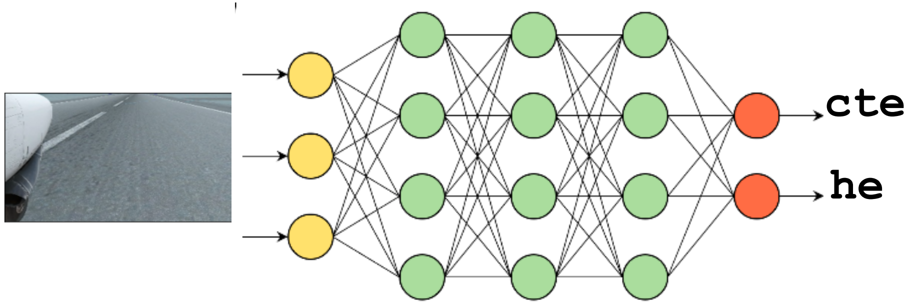
- Monitor DNN outputs
- Go to “safe fail state” if assumption is violated

Extract local properties from assumption

- More natural for DNNs
- Guide training and testing of the DNN



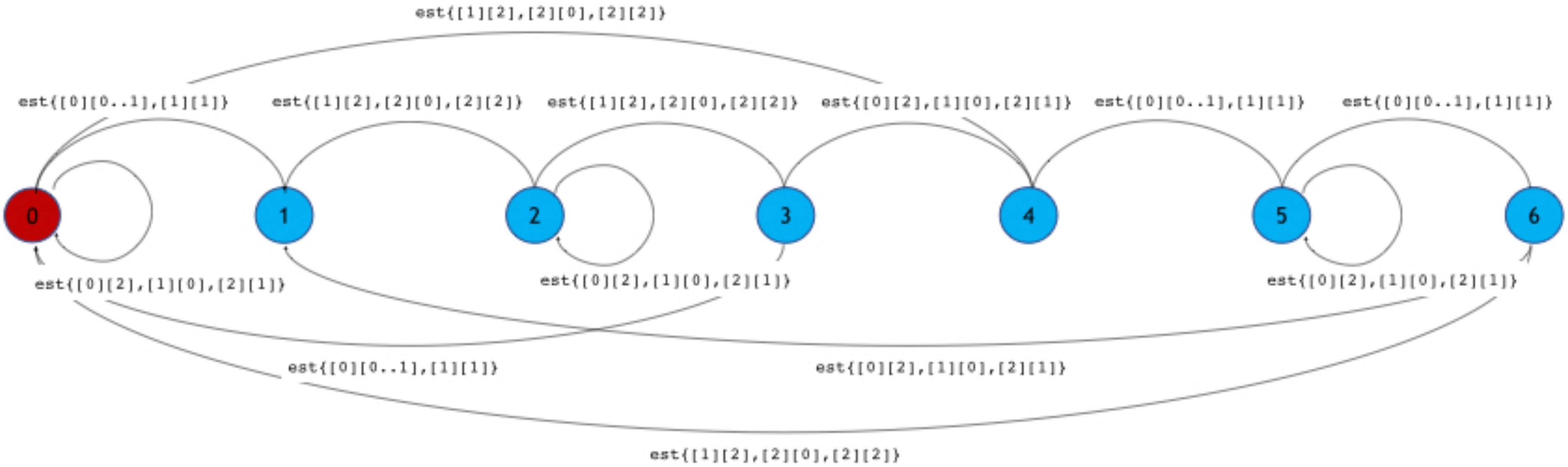
Discretized View of the DNN TaxiNet



$$\underline{\texttt{cte}} = \begin{cases} 0 \text{ if } \texttt{cte} \in [-8, -2.7) \\ 1 \text{ if } \texttt{cte} \in [-2.7, 2.7] \\ 2 \text{ if } \texttt{cte} \in (2.7, 8] \end{cases}$$

$$\underline{\texttt{he}} = \begin{cases} 1 \text{ if } \texttt{he} \in [-35, -11.67) \\ 0 \text{ if } \texttt{he} \in [-11.67, 11.66] \\ 2 \text{ if } \texttt{he} \in (11.66, 35.0] \end{cases}$$

Assumptions for Run-Time Monitoring



- Set alphabet to be only in terms of **estimates**
- Generated assumption defines allowable **temporal behavior** over the DNN outputs

Extracting Local Properties

```
Assumption_TaxiNet_Err = Q0 ,
Q0 = ( est {[0][0..1] , [1][1]} → Q1
  | est {[0][2] , [1][0] , [2][1]} → Q8
  | est {[1][2] , [2].{[0] , [2]}} → Q9 ) ,
Q1 = ( act [2][2] → Q3 ) ,
Q3 = ( est {[0][0..2] , [1][0..1] , [2][1]} → ERROR
  | est {[1][2] , [2].{[0] , [2]}} → Q4 ) ,
Q4 = ( act [2][0] → Q5 ) ,
Q5 = ( est {[0][0..1] , [1][1]} → ERROR
  | est {[0][2] , [1][0] , [2][1]} → Q4
  | est {[1][2] , [2].{[0] , [2]}} → Q6 ) ,
Q6 = ( act [1][1] → Q7 ) ,
Q7 = ( est {[1][2] , [2].{[0] , [2]}} → ERROR
  | est {[0][0..1] , [1][1]} → Q8
  | est {[0][2] , [1][0] , [2][1]} → Q9 ) ,
Q8 = ( act [1][0] → Q0 ) ,
Q9 = ( act [0][1] → Q10 ) ,
Q10 = ( est {[0][2] , [1].{[0] , [2]} , [2][0..2]} →
ERROR
  | est {[0][0..1] , [1][1]} → Q11 ) ,
Q11 = ( act [0][0] → Q12 ) ,
Q12 = ( est {[1][2] , [2].{[0] , [2]}} → ERROR
  | est {[0][2] , [1][0] , [2][1]} → Q11
  | est {[0][0..1] , [1][1]} → Q13 ) ,
Q13 = ( act [1][2] → Q14 ) ,
Q14 = ( est {[0][0..1] , [1][1]} → ERROR
  | est {[0][2] , [1][0] , [2][1]} → Q1
  | est {[1][2] , [2].{[0] , [2]}} → Q8 ) .
```

Assumption only restricts incorrect DNN behavior!
When actuals are [2][2], estimates [0][0..2] , [1][0..1] , [2][1] lead to error.

Local Properties

"When actuals are [2][2], estimates [0][0..2] , [1][0..1] , [2][1] lead to error."

$$(cte_* \in [2.7, 8) \wedge he_* \in (11.66, 35.0]) \Rightarrow ((cte \in [-2.7, 2.7] \wedge he \in (11.66, 35.0]) \vee (cte \in [2.7, 8) \wedge he \in [-11.67, 11.66]) \vee (cte \in [2.7, 8) \wedge he \in (11.66, 35.0]))$$

cte_*, he_* = actuals
cte , he = estimates

- Extracted local properties tolerate some output values that are different than the ground truth, as they don't affect safety of the overall system
- Could be used for DNN testing and training; relaxed training objective allows increased flexibility during training
- DNN verification?

Evaluation

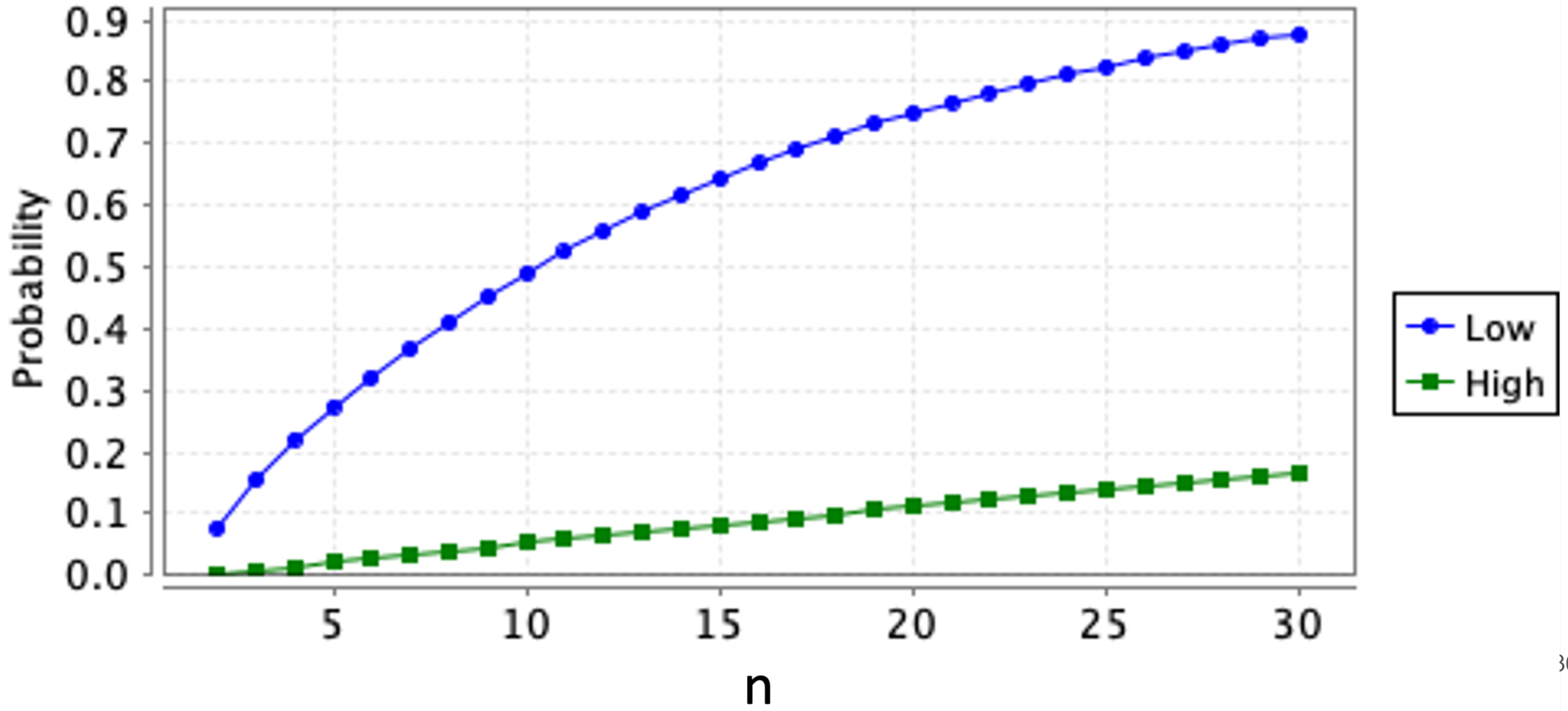
- Scalability
  - Assumptions for increasing alphabet sizes (increasing number of DNN outputs)
  - Used LTSA tool
- Permissiveness of run-time monitor
  - Run-time monitor blocks system when assumption is violated
  - Safe but prevents the system to operate
  - Used Prism to compute probability of assumption violation for two DNN models (high vs low accuracy)

Assumptions for Increasing Alphabet Sizes

MaxCTE	Assump. size	M_1 size	Time (sec.)	Memory (KB)
2	7	99	0.079	9799
4	13	261	0.126	10556
6	19	495	0.098	9926
14	43	2151	0.143	13324
30	91	8919	0.397	31056
50	151	23859	2.919	45225
100	301	92709	81.529	132418

Our approach can handle DNN classifiers with **hundreds** of of outputs

Probability of Assumption Violation



Summary

- Presented worst-case analysis approach for autonomous systems with DNN-based perception
- Generated “weakest assumptions” on DNN behavior that **guarantee** safety properties
- Can be used as run-time monitors
- Extracted local specifications on DNN behavior; can be used for training and testing

Future work:

- Systems with multiple perception components (camera and LIDAR)
  - Decompose global assumption into component-wise assumptions
- Incremental techniques for assumption generation
- Neuro-symbolic techniques for DNN training, as guided by assumptions and local properties
- Assumptions for LLMs?

Related Work

- Proving safety properties of autonomous systems with low-dimensional sensor readings
  - *Ivanov, R., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verisig: verifying safety properties of hybrid systems with neural network controllers. (2019)*
  - *Ivanov, R., Jothimurugan, K., Hsu, S., Vaidya, S., Alur, R., Bastani, O.: Compositional learning and verification of neural network controllers. (2021)*
  - Intractable for systems that use rich sensors producing high-dimensional inputs such as images
- More closely related works build models based on the analysis of the perception components
  - *Katz, S.M., Corso, A.L., Strong, C.A., Kochenderfer, M.J.: Verification of image-based neural network controllers using generative models. (2022)*
  - *Shoukry, Y.: Nnlander-verif: A neural network formal verification framework for vision-based autonomous aircraft landing. (2022)*
  - *P,H.,Deka,N.,D'Souza,D.,Lodaya,K.,Prabhakar,P.:Verification of camera-based autonomous systems.(2023)*
  - They either do not provide guarantees or do not scale to large networks
- Falsification techniques
  - *Dreossi, T., Donzé, A., Seshia, S.A.: Compositional falsification of cyber-physical systems with machine learning components. (2019)*
  - *Ghosh, S., Pant, Y.V., Ravanbakhsh, H., Seshia, S.A.: Counterexample-guided synthesis of perception models and control. (2021)*
  - They do not provide guarantees
- Most closely related approach
  - *Hsieh, C., Li, Y., Sun, D., Joshi, K., Misailovic, S., Mitra, S.: Verifying controllers with vision-based perception using safe approximate abstractions. (2022)*
  - Builds abstractions of the DNN components as guided by system-level safety properties.
  - Does not provide strong system-level guarantees
  - Provides a probabilistic result that measures empirically how close a real DNN is to the abstraction
- Probabilistic verification
  - *Incer, I., Badithela, A., Graebener, J., Mallozzi, P., Pandey, A., Yu, S.J., Benveniste, A., Caillaud, B., Murray, R.M., Sangiovanni-Vincentelli, A., et al.: Pacti: Scaling assume-guarantee reasoning for system analysis and design. (2023)*
  - They either do not incorporate DNN-specific analysis
- Safe shielding
  - *Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., Topcu, U.: Safe reinforcement learning via shielding (2017)*
  - Does not consider complex DNN
  - Our assumptions monitor DNN outputs instead of controller actions (as in shielding); can prevent errors earlier
  - Further, local specifications enable DNN testing and training

Thank you!