Algorithmic Improvisation for Dependable and Secure Autonomy

Sanjit A. Seshia

Professor

EECS Department, UC Berkeley

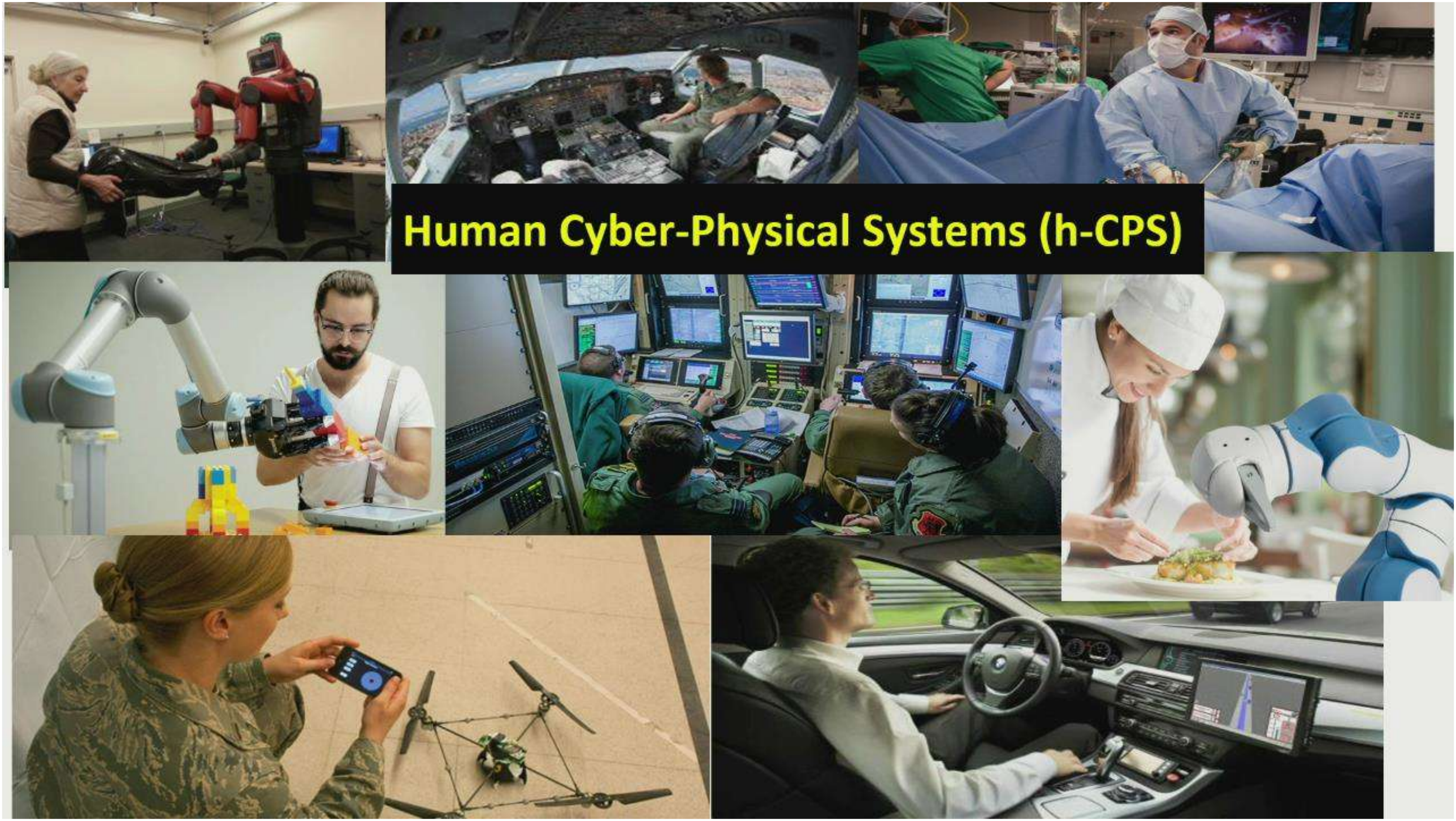
Joint work with:

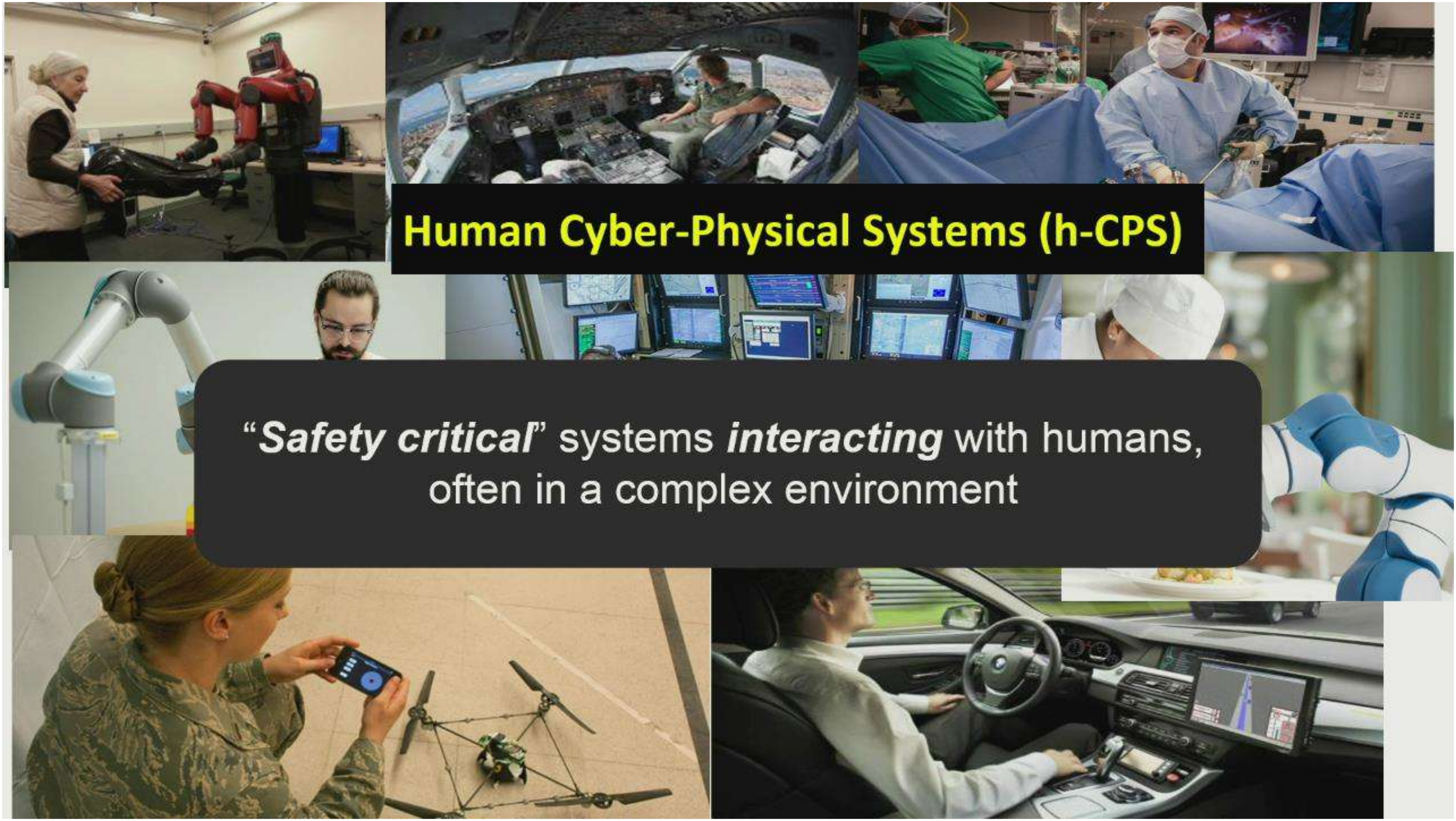
Daniel Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Ilge Akkaya, Alexandre Donze, Rafael Valle, Edward A. Lee, Alberto Sangiovanni-Vincentelli, David Wessel

VeriCal

Microsoft Research India

August 1, 2019


Human Cyber-Physical Systems (h-CPS)



Human Cyber-Physical Systems (h-CPS)

"***Safety critical***" systems ***interacting*** with humans, often in a complex environment

[NTSB]

ars TECHNICA

BIZ & IT TECH SCIENCE POLICY CARS GAMING & CULTURE

DRIVERLESS CAR SAFETY —

Report: Software bug led to death in Uber's self-driving crash

Sensors detected Elaine Herzberg, but software reportedly decided to ignore her.

TIMOTHY B. LEE - 5/7/2018, 3:12 PM

[NTSB]

Investigators with the federal agency determined that the car's detection systems, including radar and laser instruments, observed a woman walking her bicycle across the road roughly six seconds before impact — likely enough time, in other words, for a vehicle driving 43 mph to brake and avoid fatally injuring the woman.

But it did not immediately identify the woman as a human pedestrian. Instead, the agency said, "as the vehicle and pedestrian paths converged, the self-driving system software classified the pedestrian as an unknown object, as a vehicle, and then as a bicycle with varying expectations of future travel path."

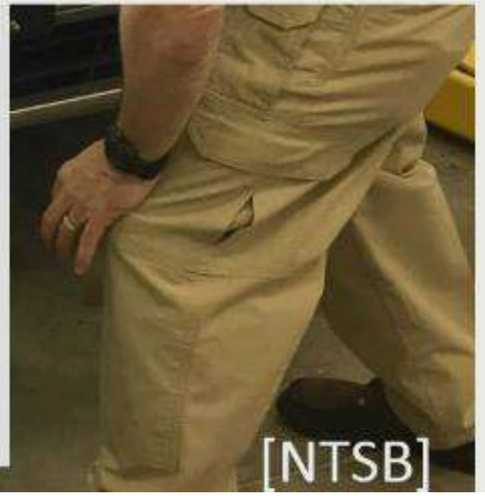
ars TECHNICA BIZ

DRIVERLESS CAR SAFETY —

Report: Software bug led to death in Uber's self-driving crash

Sensors detected Elaine Herzberg, but software reportedly decided to ignore her.

TIMOTHY B. LEE - 5/7/2018, 3:12 PM



A Formal Methods Approach

Formal Methods: rigorous algorithmic techniques to model, design, and analyze systems based on formal mathematical reasoning.

"Towards Verified Artificial Intelligence", S. A. Seshia et al., 2016:
https://arxiv.org/abs/1606.08514

A Formal Methods Approach

Formal Methods: rigorous algorithmic techniques to model, design, and analyze systems based on formal mathematical reasoning.

Prove rigorous guarantees when possible;

"Towards Verified Artificial Intelligence", S. A. Seshia et al., 2016:
https://arxiv.org/abs/1606.08514

A Formal Methods Approach

Formal Methods: rigorous algorithmic techniques to model, design, and analyze systems based on formal mathematical reasoning.

Prove rigorous guarantees when possible;

do ***formally-guided, intelligent simulation/testing*** when not.

"Towards Verified Artificial Intelligence", S. A. Seshia et al., 2016:
https://arxiv.org/abs/1606.08514

Message of this Talk

Message of this Talk

Randomness can play a crucial role in Formal Methods for Autonomy!

Message of this Talk

Randomness can play a crucial role in Formal Methods for Autonomy!

- Modeling Autonomous Systems and Their Environments

Message of this Talk

Randomness can play a crucial role in Formal Methods for Autonomy!

- Modeling Autonomous Systems and Their Environments
- Verification – Constrained-Random Simulation, Probabilistic Verification

Message of this Talk

Randomness can play a crucial role in Formal Methods for Autonomy!

- Modeling Autonomous Systems and Their Environments
- Verification – Constrained-Random Simulation, Probabilistic Verification
- Synthesis: Randomized Planning/Control

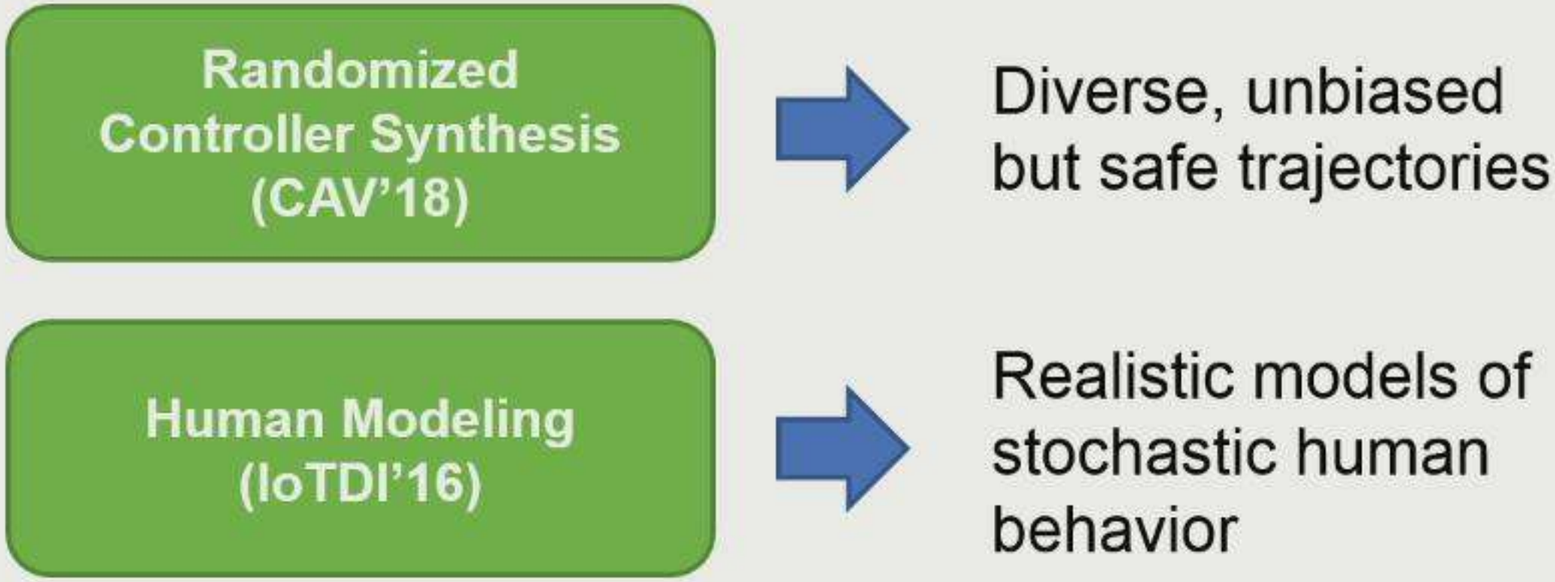
Randomized Formal Methods for Safe Autonomy

Randomized Formal Methods for Safe Autonomy

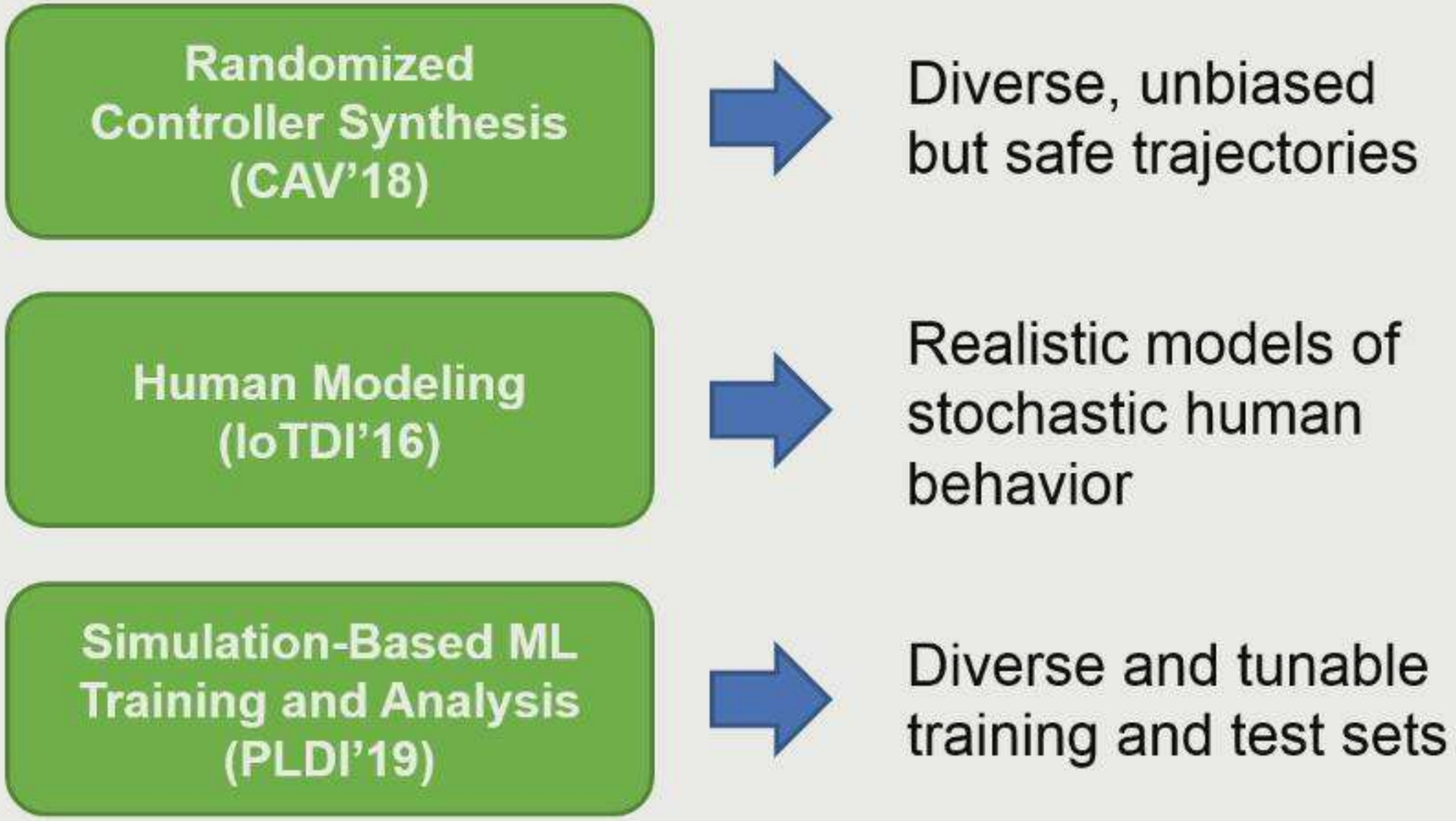
Randomized Controller Synthesis (CAV'18)

Diverse, unbiased but safe trajectories

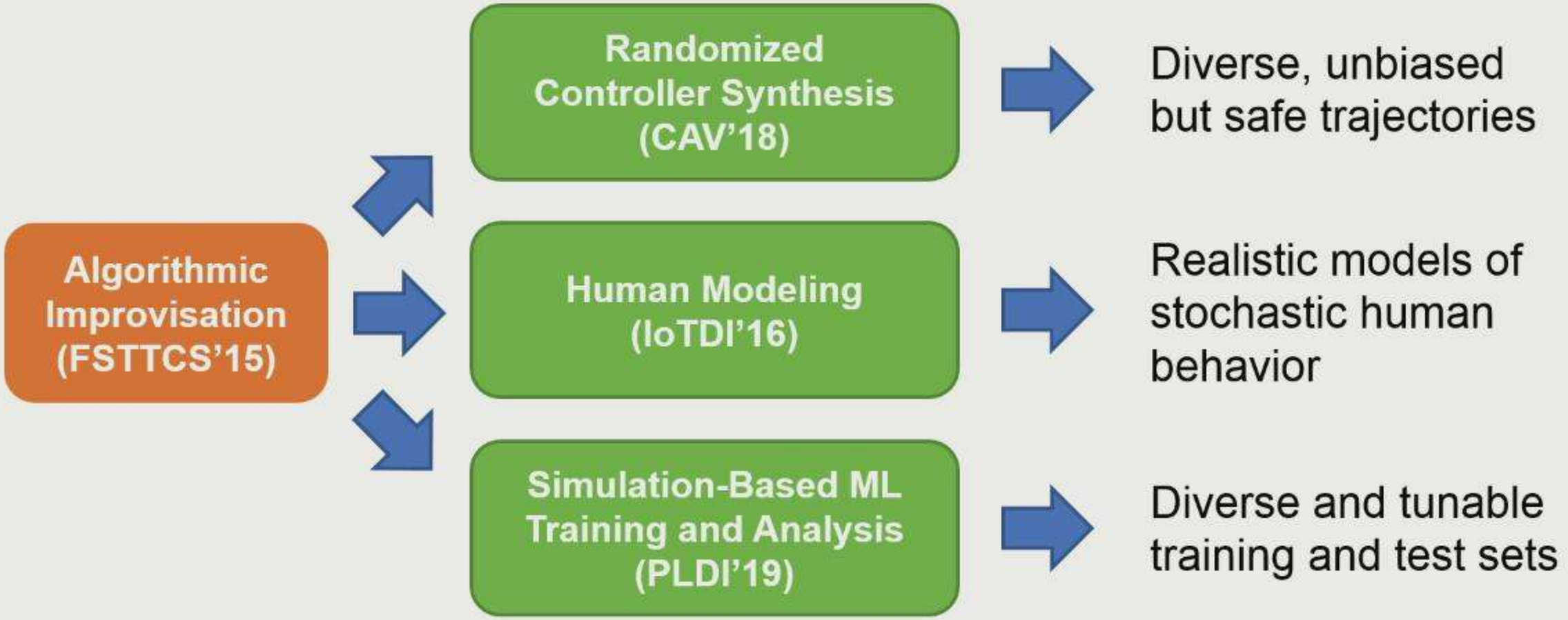
Randomized Formal Methods for Safe Autonomy



Randomized Formal Methods for Safe Autonomy



Randomized Formal Methods for Safe Autonomy



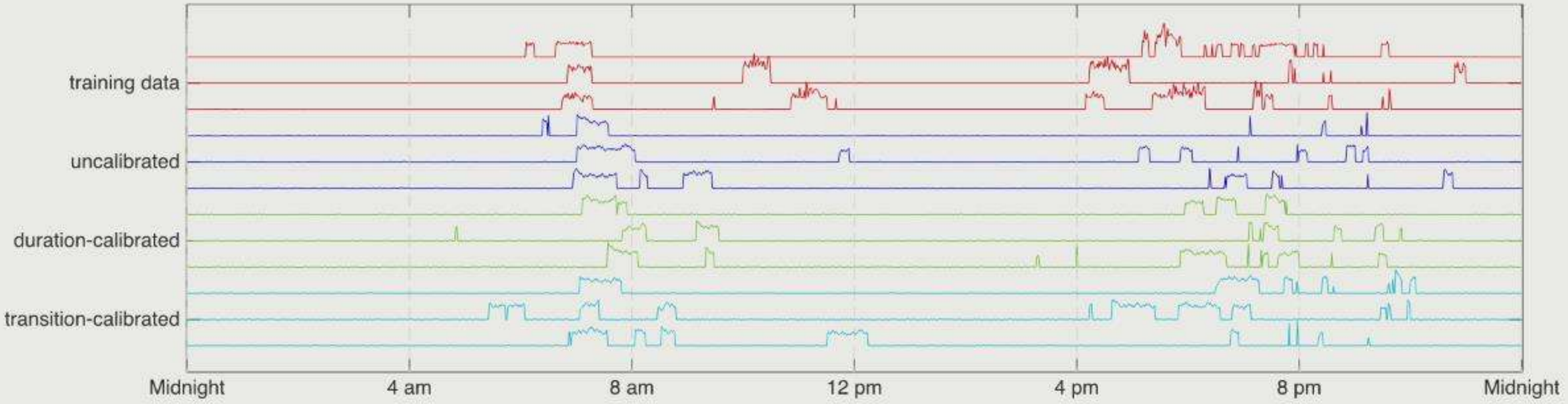
Why Randomness: Variety/Coverage

- Train/test set generator for an object-detecting neural network
  - "Generate images of bumper-to-bumper traffic"
  - Huge and diverse space of possibilities – randomness lets you cover it well

Fremont et al., *Scenic: A Language for Scenario Specification and Scene Generation*, PLDI 2019 (to appear).

Why Randomness: Realism

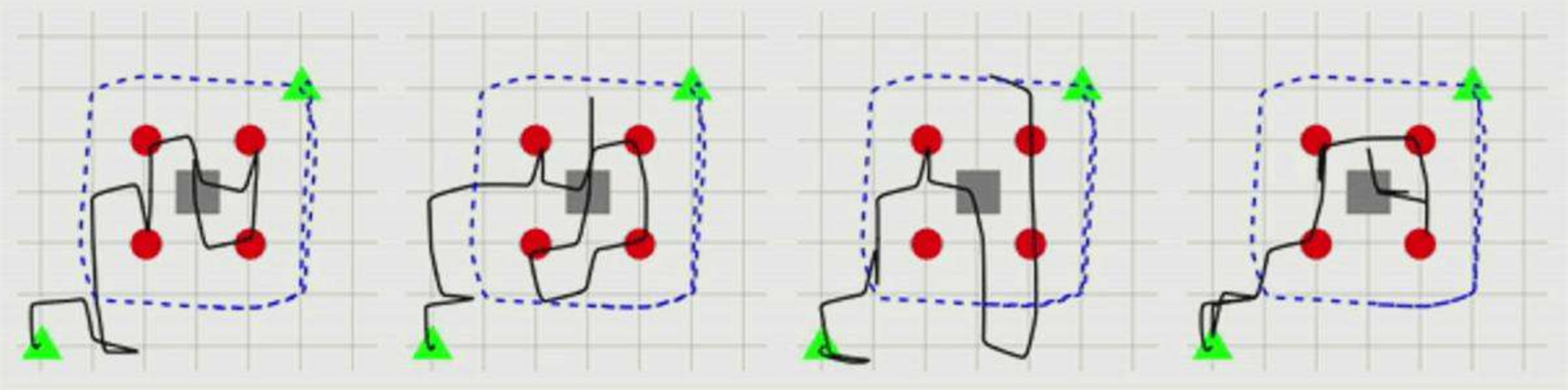
- Home automation lighting controller
  - "Mimic the user's typical behavior while they are away"
  - Deterministic models unrealistic – human behavior is random



Akkaya et al., *Control Improvisation with Probabilistic Temporal Specifications*, IoTDI 2016.

Why Randomness: Unpredictability

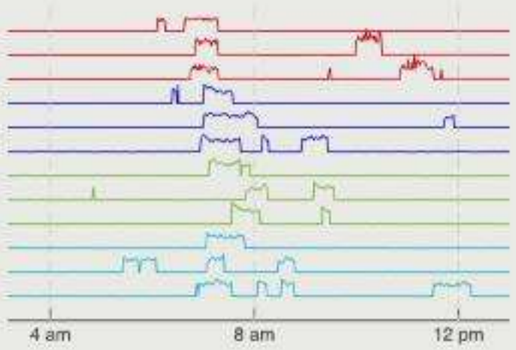
- Surveillance robot
  - “Patrol an area, visiting each important location sufficiently often”
  - Using a random route makes the robot’s future location harder to predict

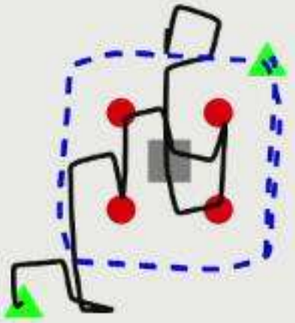


Fremont and Seshia, *Reactive Control Improvisation*, CAV 2018.

Randomness with Guarantees

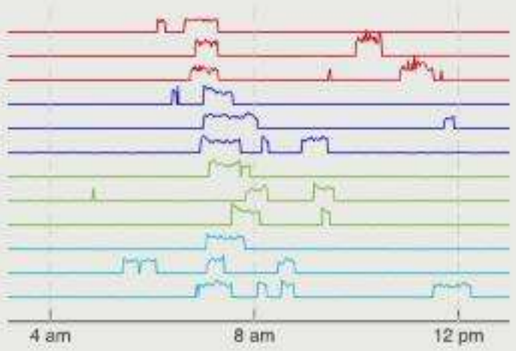
- Randomness is essential for all these systems

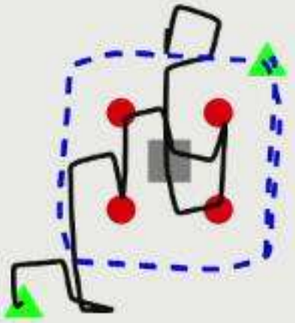




Randomness with Guarantees

- Randomness is essential for all these systems

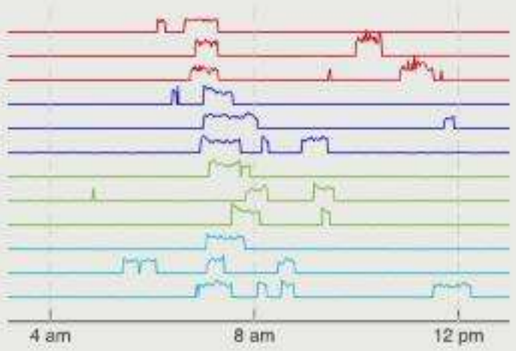


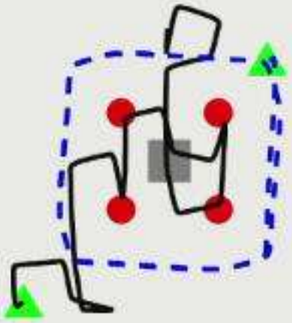


- *But*, they must also satisfy formal specifications:

Randomness with Guarantees

- Randomness is essential for all these systems



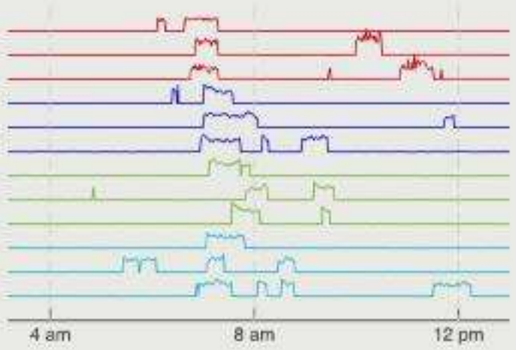


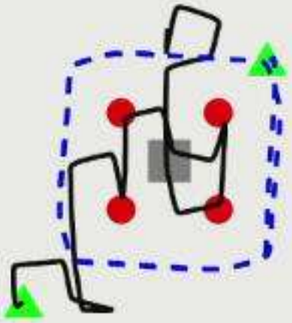
- *But*, they must also satisfy formal specifications:

"use < 20 kWh of power per day"

Randomness with Guarantees

- Randomness is essential for all these systems





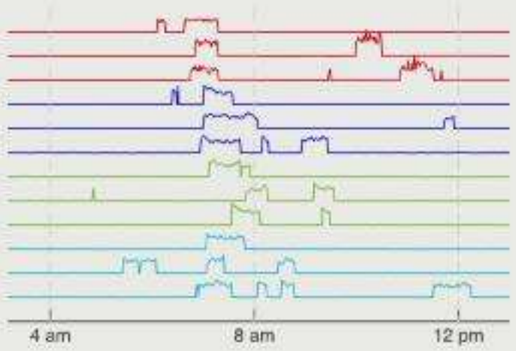
- *But*, they must also satisfy formal specifications:

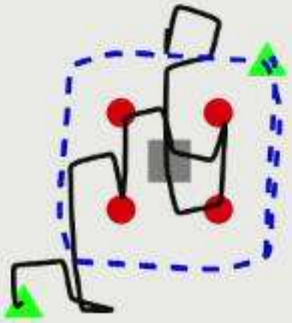
"use < 20 kWh of power per day"

"objects must not intersect"

Randomness with Guarantees

- Randomness is essential for all these systems





- *But*, they must also satisfy formal specifications:

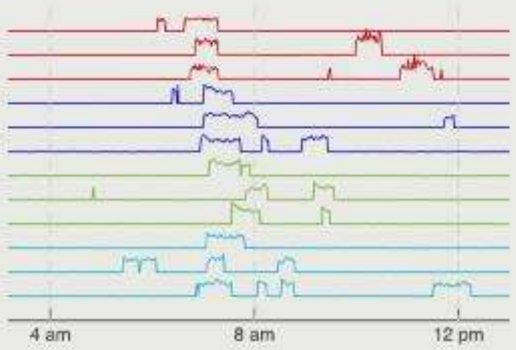
"use < 20 kWh of power per day"

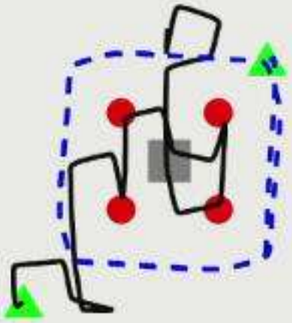
"objects must not intersect"

"never collide with another drone"

Randomness with Guarantees

- Randomness is essential for all these systems





- *But*, they must also satisfy formal specifications:

"use < 20 kWh of power per day"

"objects must not intersect"

"never collide with another drone"

- ***How to design a system with <u>random but controlled</u> behavior?***

Algorithmic Improvisation (Control Improvisation)

- A framework for automatically synthesizing (reactive) systems with random behavior but formal guarantees:

Algorithmic Improvisation (Control Improvisation)

- A framework for automatically synthesizing (reactive) systems with random behavior but formal guarantees:
  - Guaranteed safety (hard and soft constraints)

Algorithmic Improvisation (Control Improvisation)

- A framework for automatically synthesizing (reactive) systems with random behavior but formal guarantees:
  - Guaranteed safety (hard and soft constraints)
  - Guaranteed randomness (randomness constraint)

Algorithmic Improvisation (Control Improvisation)

- A framework for automatically synthesizing (reactive) systems with random behavior but formal guarantees:
  - Guaranteed safety (hard and soft constraints)
  - Guaranteed randomness (randomness constraint)

- **Key novelty: Randomness is *part* of the specification!**

Outline

1. Control Improvisation
   - Definition and motivating applications

Outline

1. Control Improvisation
   - Definition and motivating applications
2. Theory of CI
   - Efficient algorithms and hardness results

Outline

1. Control Improvisation
   - Definition and motivating applications
2. Theory of CI
   - Efficient algorithms and hardness results
3. Designing and Analyzing Perception Systems

Outline

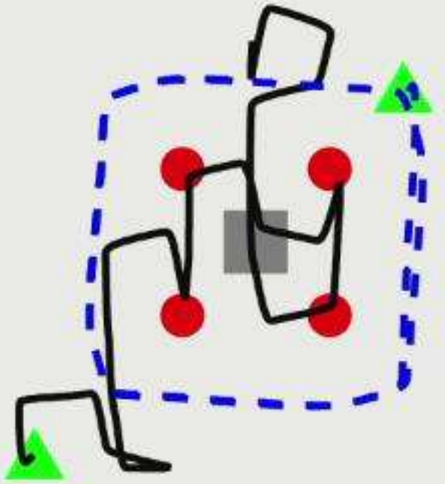
1. Control Improvisation
   - Definition and motivating applications
2. Theory of CI
   - Efficient algorithms and hardness results
3. Designing and Analyzing Perception Systems
4. Conclusion & Future Work

Outline

1. **Control Improvisation**
   - Definition and motivating applications
2. Theory of CI
   - Efficient algorithms and hardness results
3. Designing and Analyzing Perception Systems
4. Conclusion & Future Work

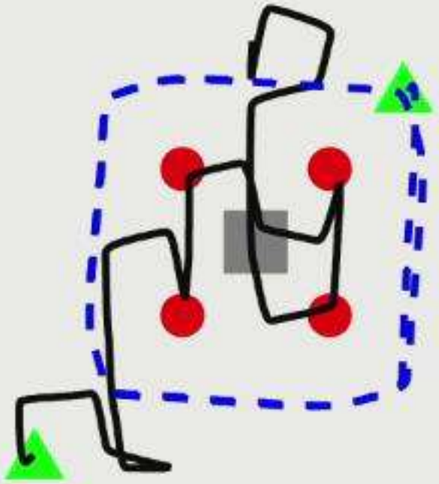
Motivating Applications

- **Robotic Surveillance**
  - “patrol an area in an unpredictable way”



Motivating Applications

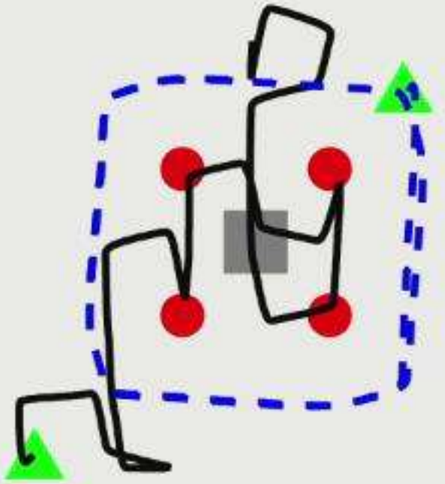
- **Robotic Surveillance**
  - “patrol an area in an unpredictable way”



- Visit each location sufficiently often

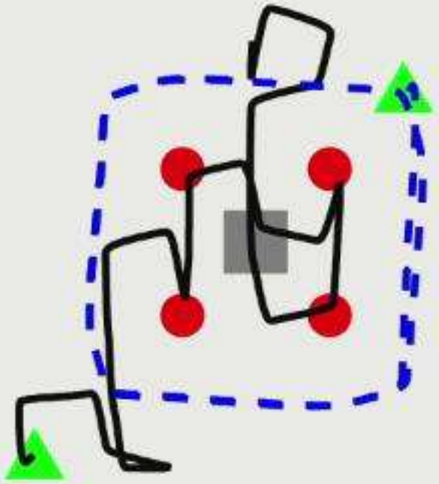
Motivating Applications

- **Robotic Surveillance**
  - “patrol an area in an unpredictable way”



Motivating Applications

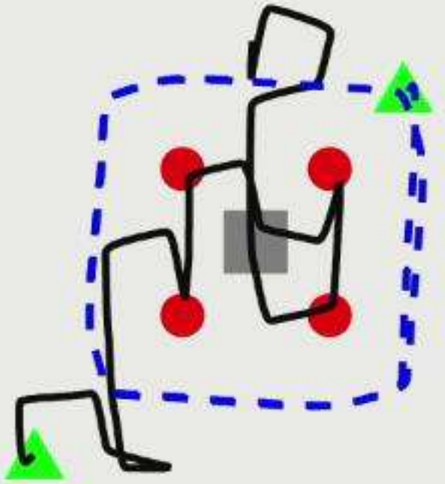
- **Robotic Surveillance**
  - “patrol an area in an unpredictable way”



- Visit each location sufficiently often

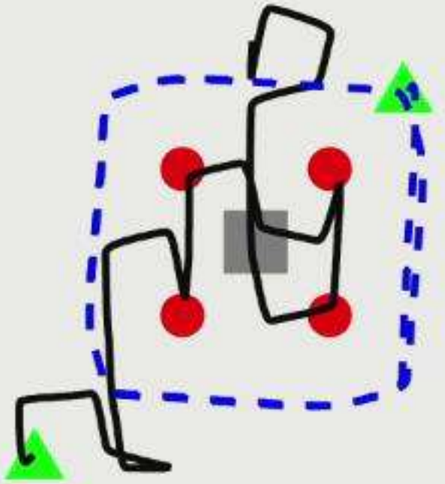
Motivating Applications

- **Robotic Surveillance**
  - “patrol an area in an unpredictable way”
- Visit each location sufficiently often
- Take a route close to the shortest one



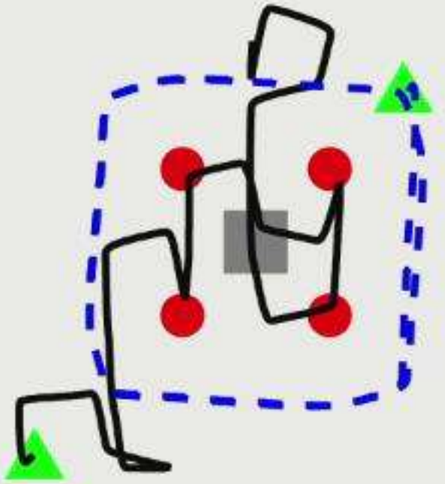
Motivating Applications

- **Robotic Surveillance**
  - “patrol an area in an unpredictable way”
- Visit each location sufficiently often
- Take a route close to the shortest one
- Don’t always take the same route



Motivating Applications

- **Robotic Surveillance**
  - “patrol an area in an unpredictable way”



- Visit each location sufficiently often

Hard Constraint

- Take a route close to the shortest one

Soft Constraint

- Don’t always take the same route

Randomness Constraint

Motivating Applications

- **Synthetic Data Generation**
  - “create traffic images for this neural network”

Motivating Applications

- **Synthetic Data Generation**
  - “create traffic images for this neural network”

- Objects should not intersect

Motivating Applications

- **Synthetic Data Generation**
  - “create traffic images for this neural network”

- Objects should not intersect

- Usually, be similar to real-world traffic

Motivating Applications

- **Synthetic Data Generation**
  - “create traffic images for this neural network”

- Objects should not intersect
- Usually, be similar to real-world traffic
- Generate a diverse set of images

Motivating Applications

- **Synthetic Data Generation**
  - “create traffic images for this neural network”

- Objects should not intersect

Hard Constraint

- Usually, be similar to real-world traffic

Soft Constraint

- Generate a diverse set of images

Randomness Constraint

Commonalities

- These and other applications: music improvisation, fuzz testing…

Generate sequences subject to three kinds of constraints:

[D. Fremont et al., “Control Improvisation”, FSTTCS 2015. Extended version on arxiv, 2018.]

Commonalities

- These and other applications: music improvisation, fuzz testing…

Generate sequences subject to three kinds of constraints:

- Hard constraint: *every* sequence satisfies some property

[D. Fremont et al., “Control Improvisation”, FSTTCS 2015. Extended version on arxiv, 2018.]

Commonalities

- These and other applications: music improvisation, fuzz testing…

Generate sequences subject to three kinds of constraints:

- Hard constraint: *every* sequence satisfies some property
- Soft constraint: *most* sequences satisfy some property

[D. Fremont et al., “Control Improvisation”, FSTTCS 2015. Extended version on arxiv, 2018.]

Commonalities

- These and other applications: music improvisation, fuzz testing…

Generate sequences subject to three kinds of constraints:

- Hard constraint: *every* sequence satisfies some property
- Soft constraint: *most* sequences satisfy some property
- Randomness constraint: no sequence is generated too frequently

[D. Fremont et al., “Control Improvisation”, FSTTCS 2015. Extended version on arxiv, 2018.]

Commonalities

- These and other applications: music improvisation, fuzz testing…

Generate sequences subject to three kinds of constraints:

- Hard constraint: *every* sequence satisfies some property
- Soft constraint: *most* sequences satisfy some property
- Randomness constraint: no sequence is generated too frequently

Control Improvisation is a precisely-defined theoretical problem capturing these requirements

[D. Fremont et al., “Control Improvisation”, FSTTCS 2015. Extended version on arxiv, 2018.]

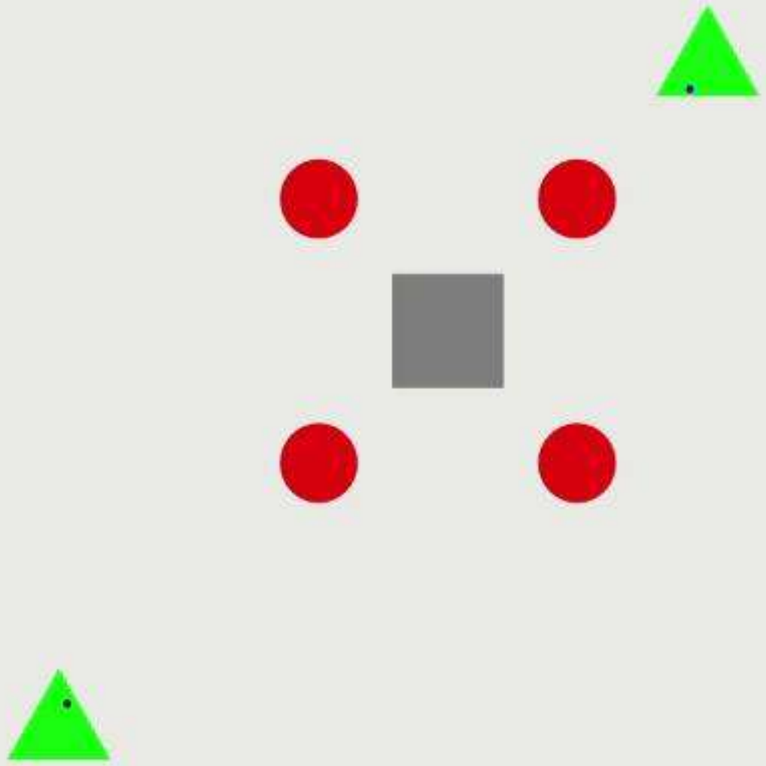
Related Work

- Control improvisation is a fundamentally new type of problem

	HARD	SOFT	RANDOM	REACTIVE
Factor Oracles [1]		■		
Mutational Fuzzers [2]		■		
Uniform Sampling [3]	■		■	
Generative Fuzzers [2,3]	■		■	
Reactive Synthesis [4]	■			■
Control Improvisation	■	■	■	■

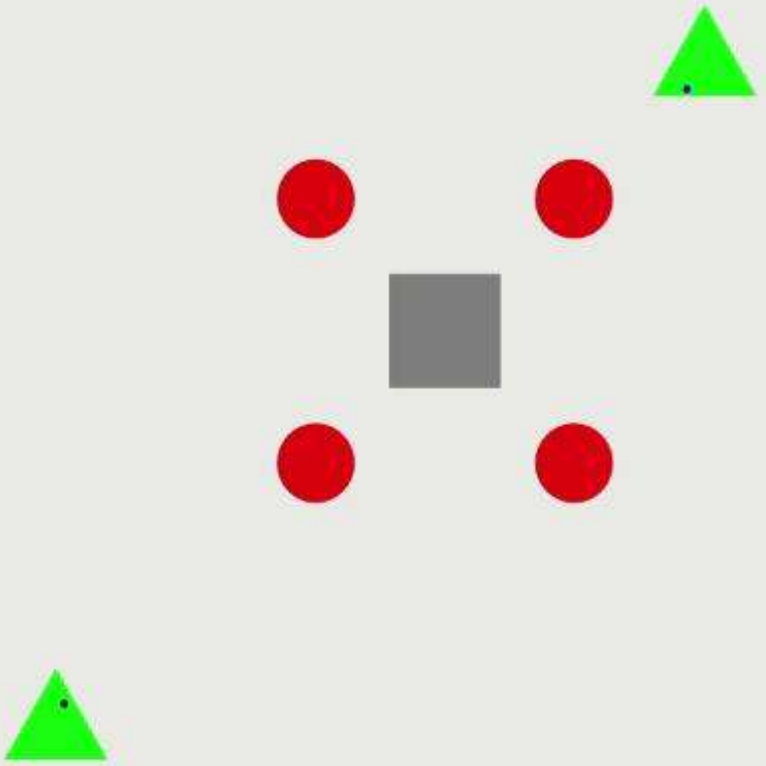
[1] Assayag & Dubnov, *Soft Computation*, 2004.
[2] see Sutton, Greene, & Amini, Addison-Wesley, 2007.
[3] e.g. Hickey & Cohen, *SIAM Journal on Computing*, 1983.
[4] e.g. Pnueli & Rosner, POPL 1989.

Components of the Definition



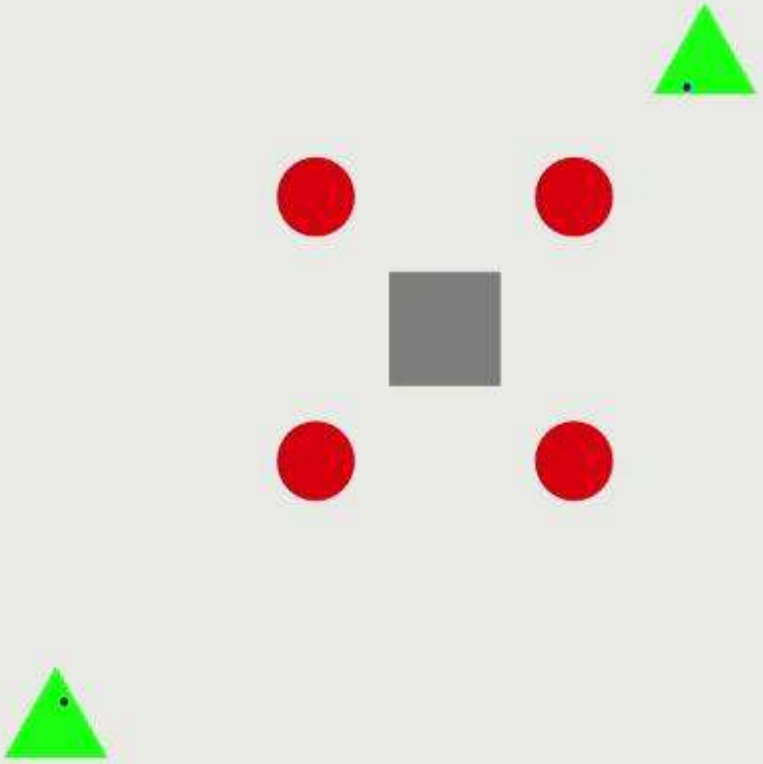
Components of the Definition

- Generate sequences of length *n* from a finite alphabet Σ
  - Σ = { N, S, E, W }, n = 30



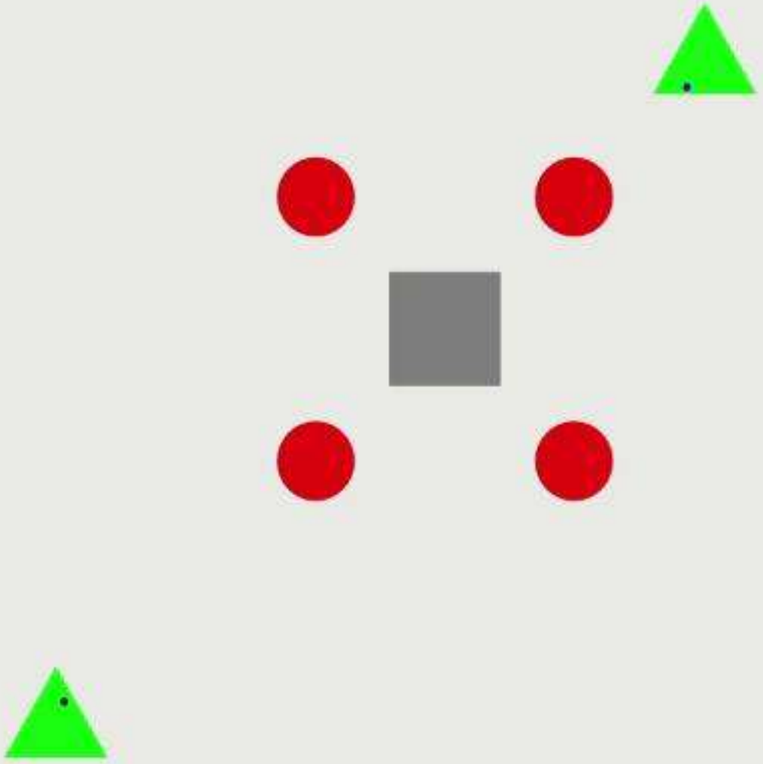
Components of the Definition

- Generate sequences of length *n* from a finite alphabet Σ
  - Σ = { N, S, E, W }, n = 30
- Hard constraints given by specification *H*
  - Visit all circles; avoid collisions

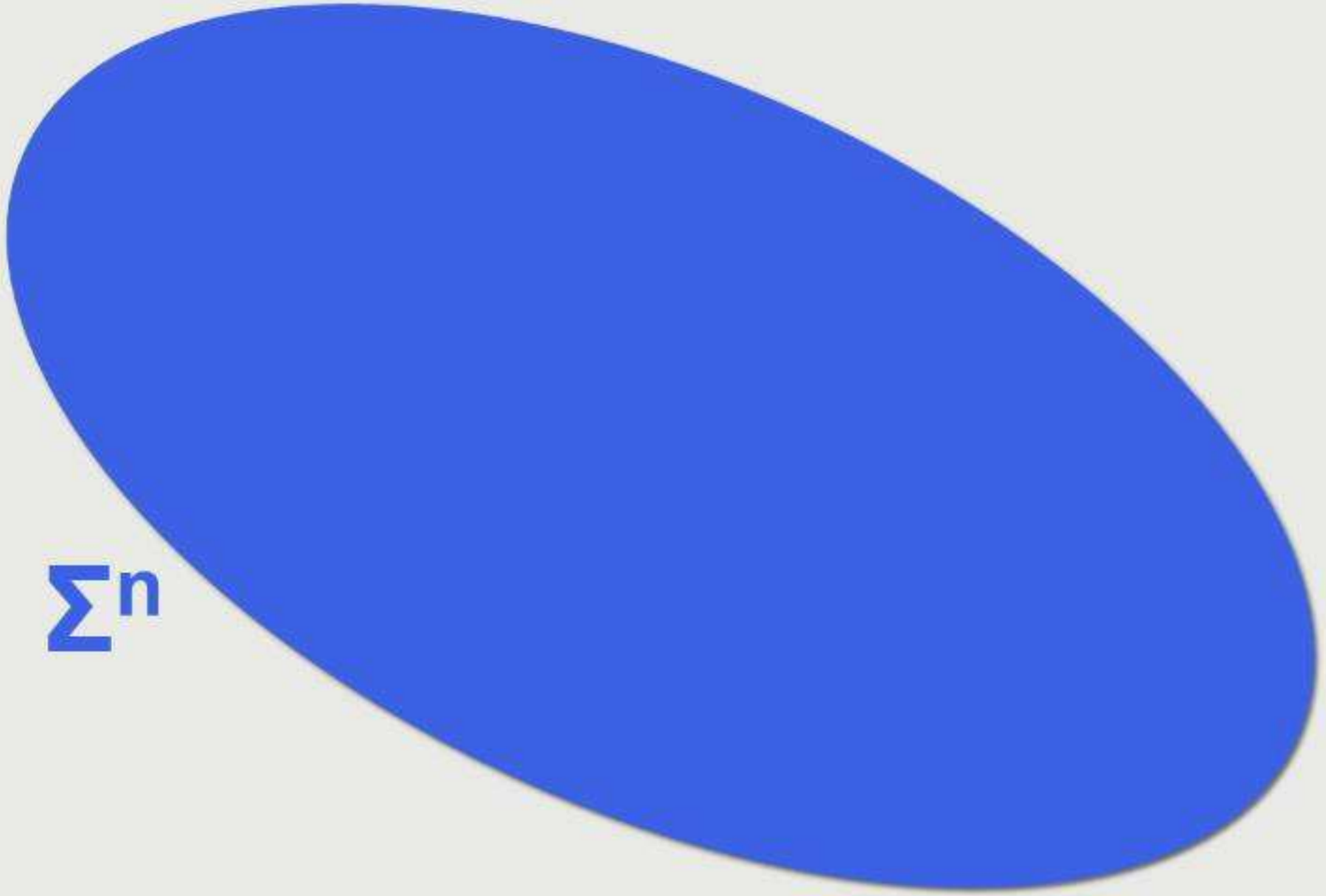


Components of the Definition

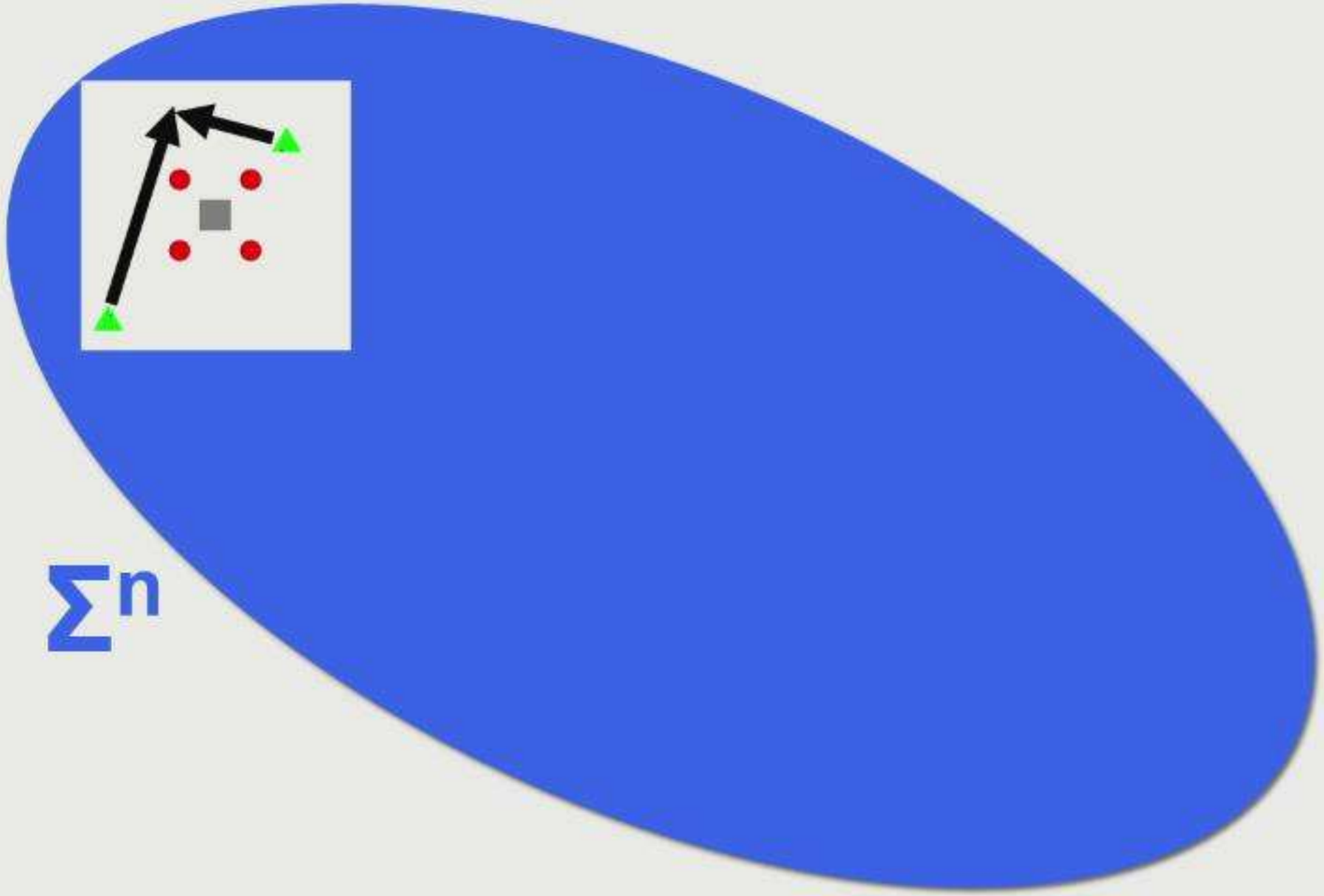
- Generate sequences of length *n* from a finite alphabet Σ
  - Σ = { N, S, E, W }, n = 30
- Hard constraints given by specification *H*
  - Visit all circles; avoid collisions
- Soft constraints given by specification *S*
  - Don't visit a circle twice



Components of the Definition

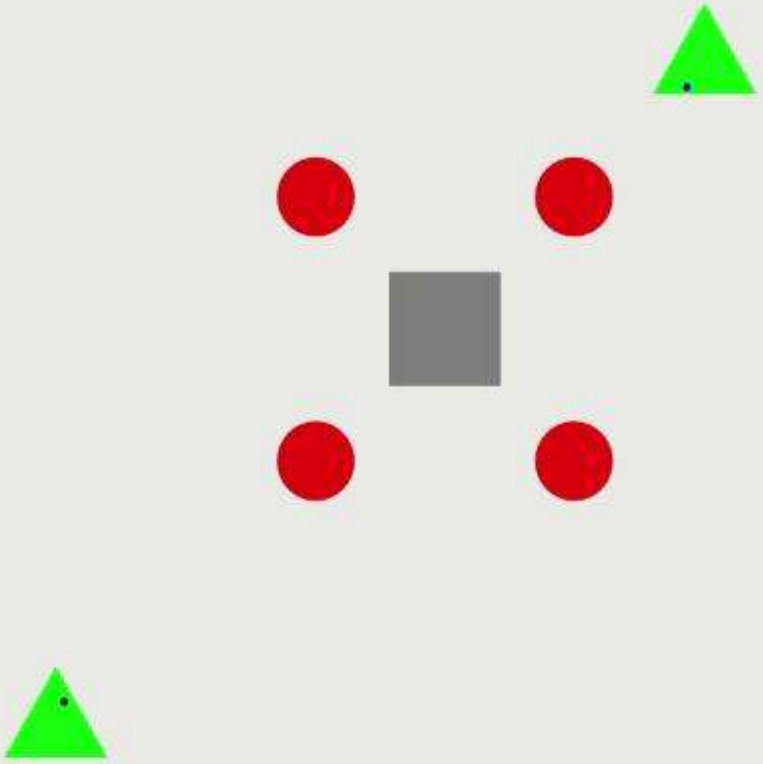


Components of the Definition

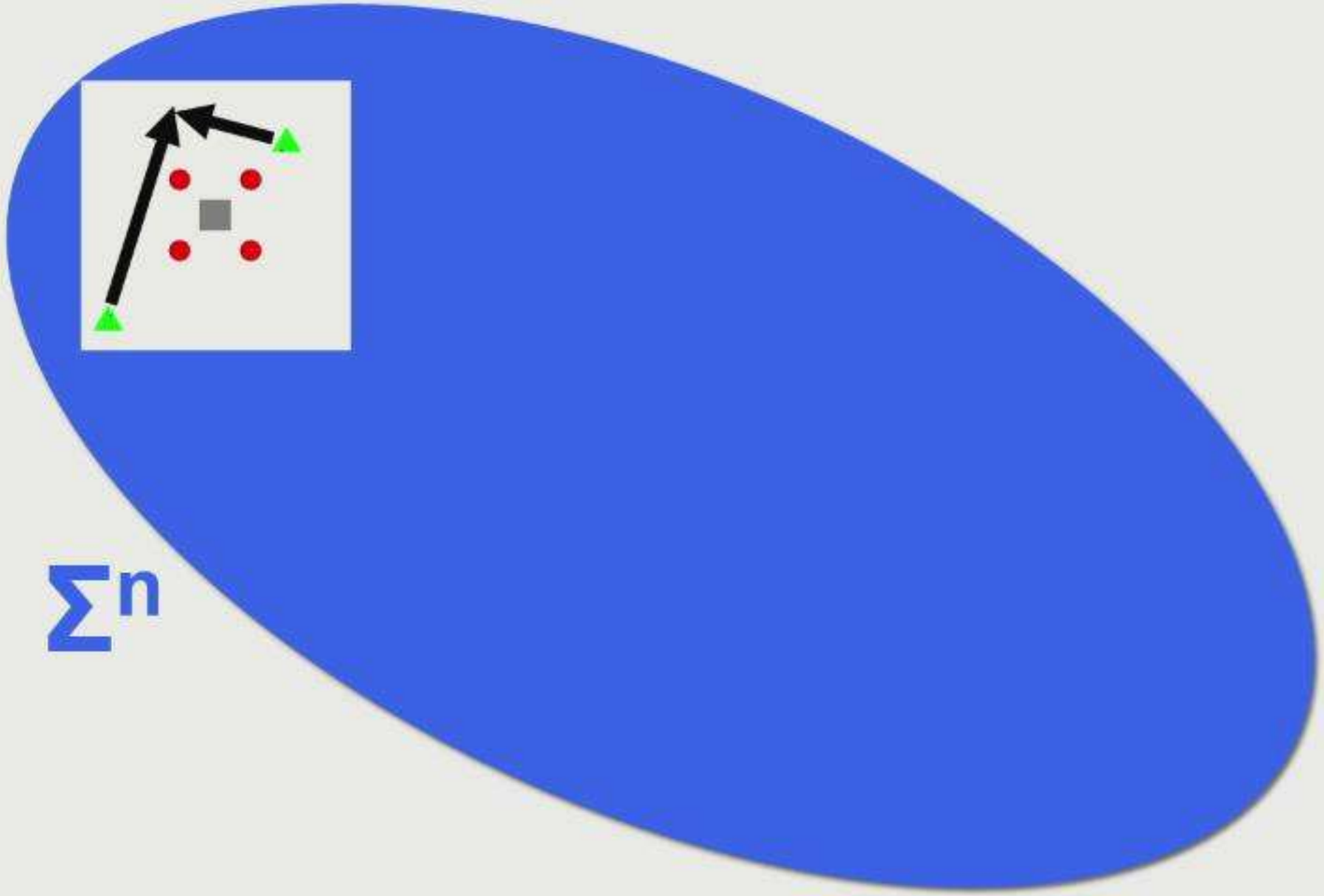


Components of the Definition

- Generate sequences of length *n* from a finite alphabet Σ
  - Σ = { N, S, E, W }, n = 30
- Hard constraints given by specification *H*
  - Visit all circles; avoid collisions
- Soft constraints given by specification *S*
  - Don't visit a circle twice

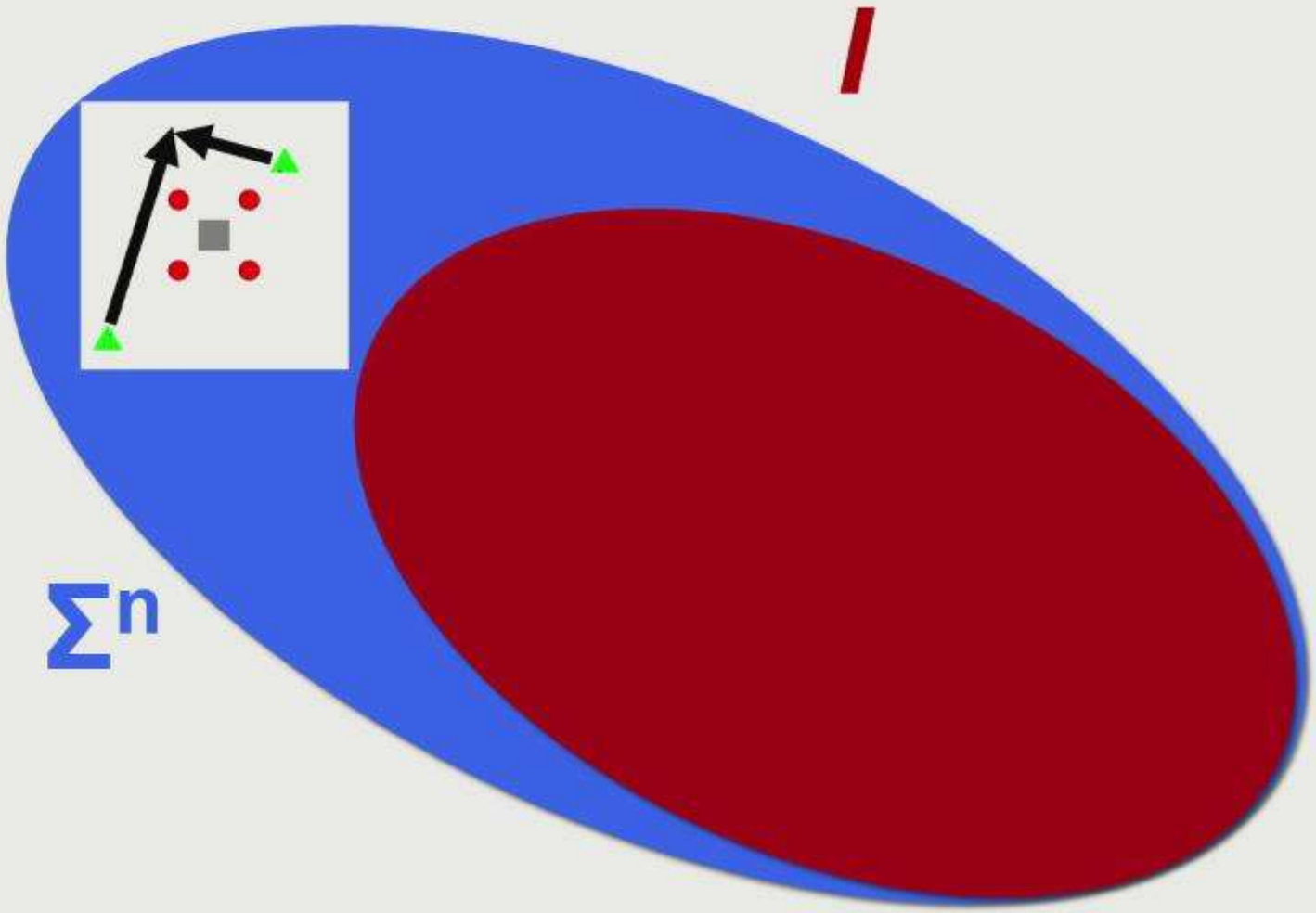


Components of the Definition



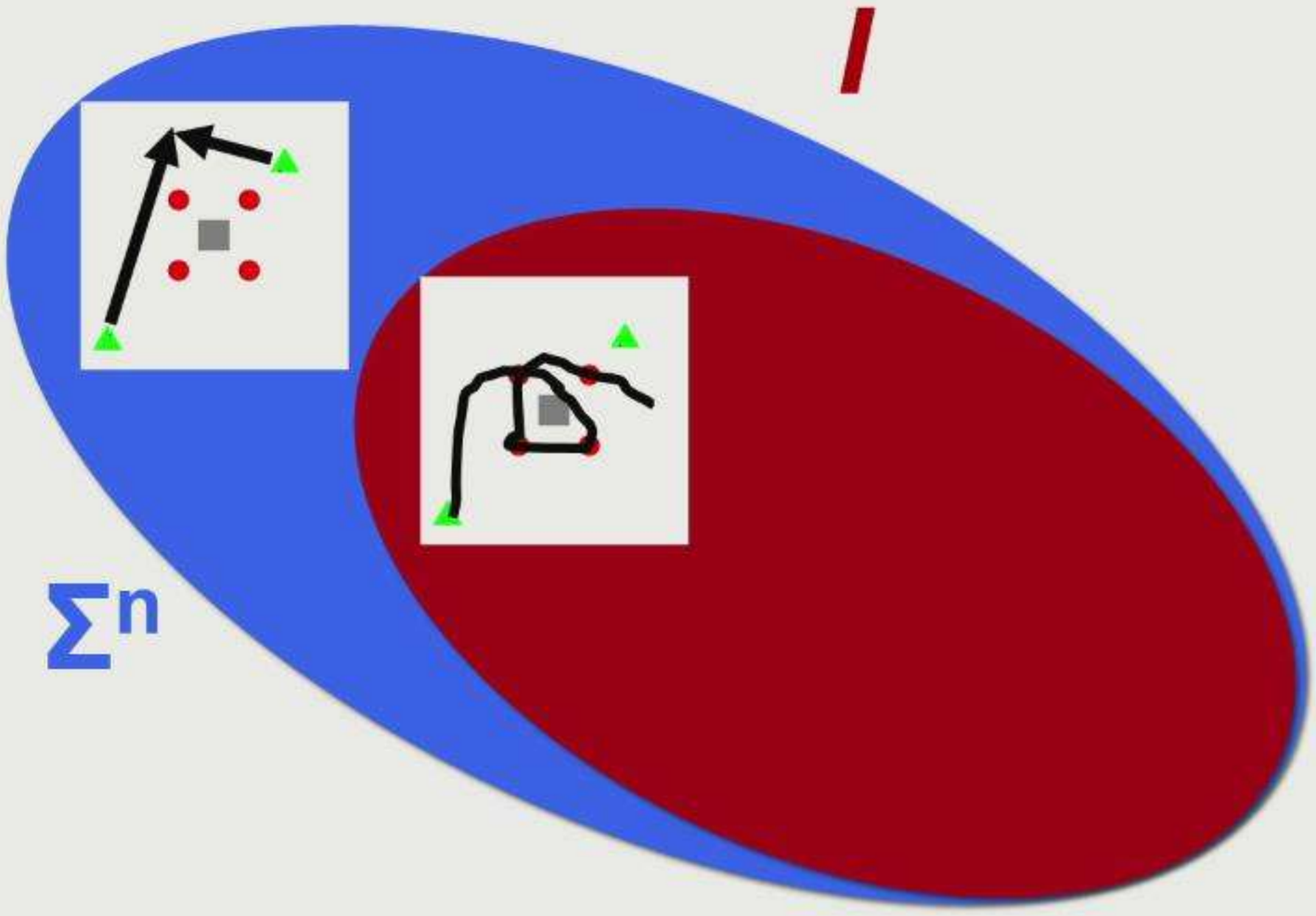
Components of the Definition

- The *improvisations* I are $L(H) \cap \Sigma^n$



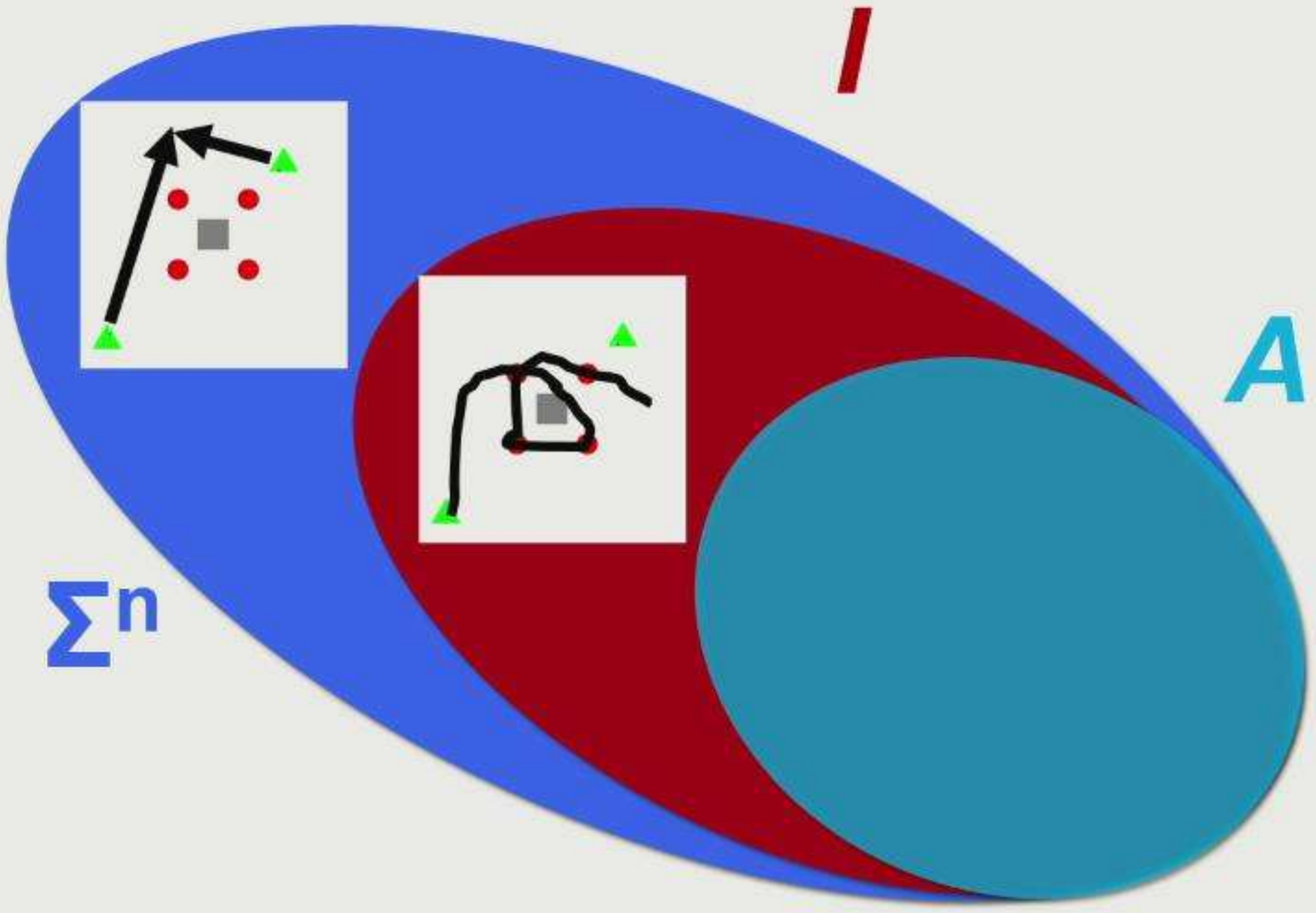
Components of the Definition

- The *improvisations* *I* are $L(H) \cap \Sigma^n$



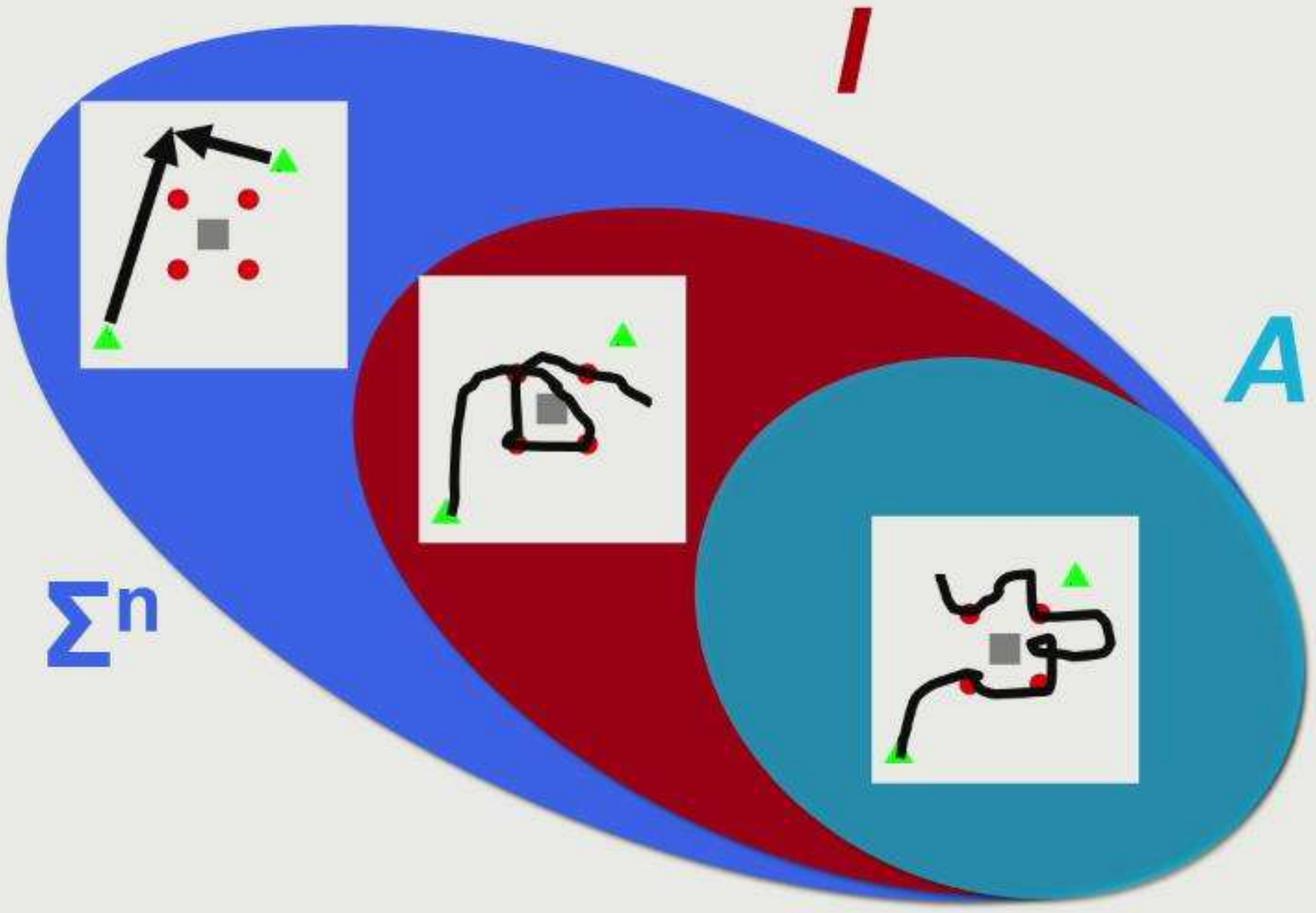
Components of the Definition

- The *improvisations I* are $L(H) \cap \Sigma^n$
- The *admissible improvisations A* are $I \cap L(S)$



Components of the Definition

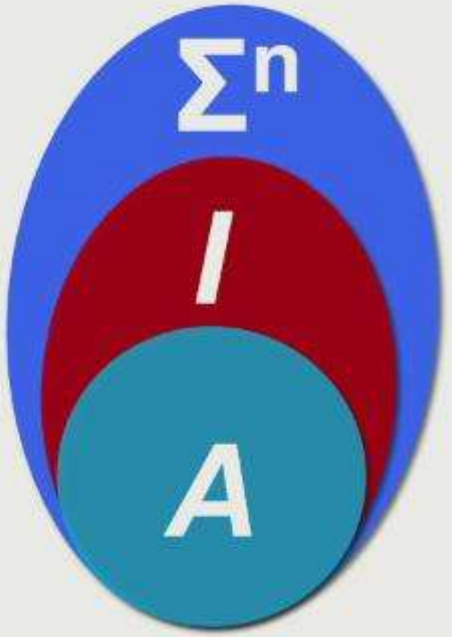
- The *improvisations* I are $L(H) \cap \Sigma^n$
- The *admissible improvisations A* are $I \cap L(S)$



Definition of Control Improviser

Given H, S, n, an *error probability* $0 \leq \varepsilon \leq 1$, and bounds $0 < \lambda, \rho \leq 1$, a distribution $D : \Sigma^* \rightarrow [0,1]$ is an *improvising distribution* if:

-
-
-

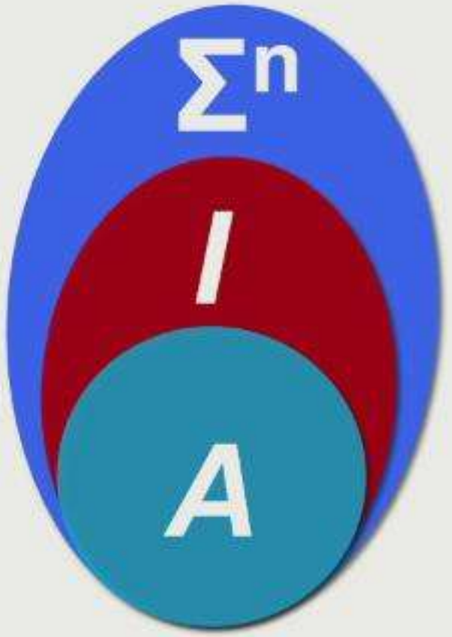


Fremont et al., *Control Improvisation*, arXiv:1704.06319, extending FSTTCS 2015.

Definition of Control Improviser

Given H, S, n, an *error probability* $0 \le \varepsilon \le 1$, and bounds $0 < \lambda, \rho \le 1$, a distribution $D : \Sigma^* \to [0,1]$ is an *improvising distribution* if:

- $\Pr[w \in I \mid w \leftarrow D] = 1$ Hard constraint
-
-

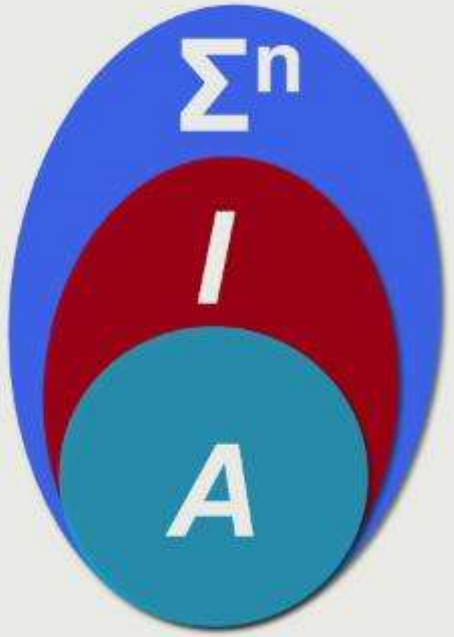


Fremont et al., *Control Improvisation*, arXiv:1704.06319, extending FSTTCS 2015.

Definition of Control Improviser

Given *H*, *S*, *n*, an *error probability* $0 \le \varepsilon \le 1$, and bounds $0 < \lambda, \rho \le 1$, a distribution $D : \Sigma^* \to [0,1]$ is an *improvising distribution* if:

- $\Pr[w \in I \mid w \leftarrow D] = 1$ Hard constraint
- $\Pr[w \in A \mid w \leftarrow D] \geq 1 - \varepsilon$ Soft constraint
-

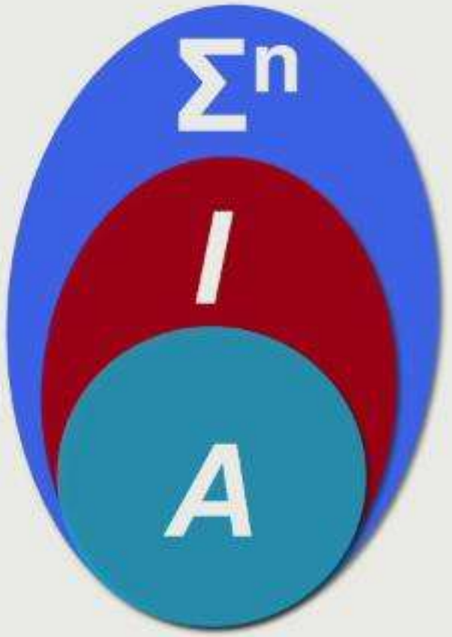


Fremont et al., *Control Improvisation*, arXiv:1704.06319, extending FSTTCS 2015.

Definition of Control Improviser

Given *H*, *S*, *n*, an *error probability* $0 \le \varepsilon \le 1$, and bounds $0 < \lambda, \rho \le 1$, a distribution $D : \Sigma^* \to [0,1]$ is an *improvising distribution* if:

- $\Pr[w \in I \mid w \leftarrow D] = 1$ Hard constraint
- $\Pr[w \in A \mid w \leftarrow D] \geq 1 - \varepsilon$ Soft constraint
- $\forall w \in I,\ \lambda \leq D(w) \leq \rho$ Randomness req.

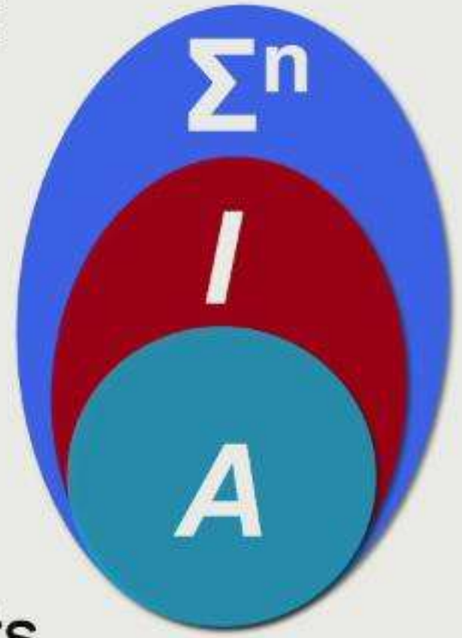


Fremont et al., *Control Improvisation*, arXiv:1704.06319, extending FSTTCS 2015.

Definition of Control Improviser

Given *H*, *S*, *n*, an *error probability* $0 \le \varepsilon \le 1$, and bounds $0 < \lambda, \rho \le 1$, a distribution $D : \Sigma^* \to [0,1]$ is an *improvising distribution* if:

- $\Pr[w \in I \mid w \leftarrow D] = 1$ — Hard constraint
- $\Pr[w \in A \mid w \leftarrow D] \geq 1 - \varepsilon$ — Soft constraint
- $\forall w \in I,\ \lambda \leq D(w) \leq \rho$ — Randomness req.



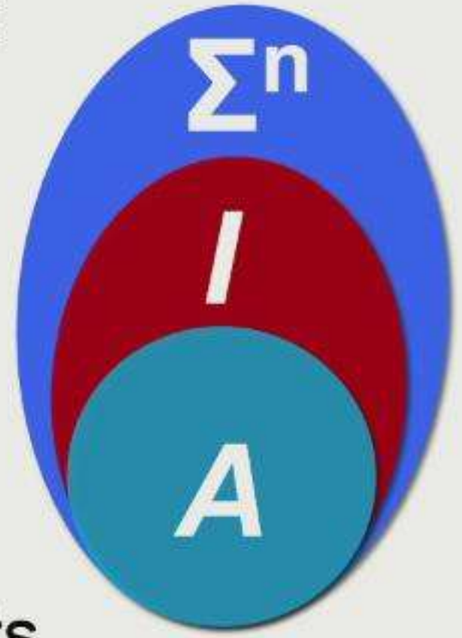
The CI instance $\mathcal{C} = (\mathcal{H}, \mathcal{S}, n, \epsilon, \lambda, \rho)$ is *feasible* if *D* exists.

Fremont et al., *Control Improvisation*, arXiv:1704.06319, extending FSTTCS 2015.

Definition of Control Improviser

Given *H*, *S*, *n*, an *error probability* $0 \le \varepsilon \le 1$, and bounds $0 < \lambda, \rho \le 1$, a distribution $D : \Sigma^* \to [0,1]$ is an *improvising distribution* if:

- $\Pr[w \in I \mid w \leftarrow D] = 1$ Hard constraint
- $\Pr[w \in A \mid w \leftarrow D] \geq 1 - \varepsilon$ Soft constraint
- $\forall w \in I,\ \lambda \leq D(w) \leq \rho$ Randomness req.



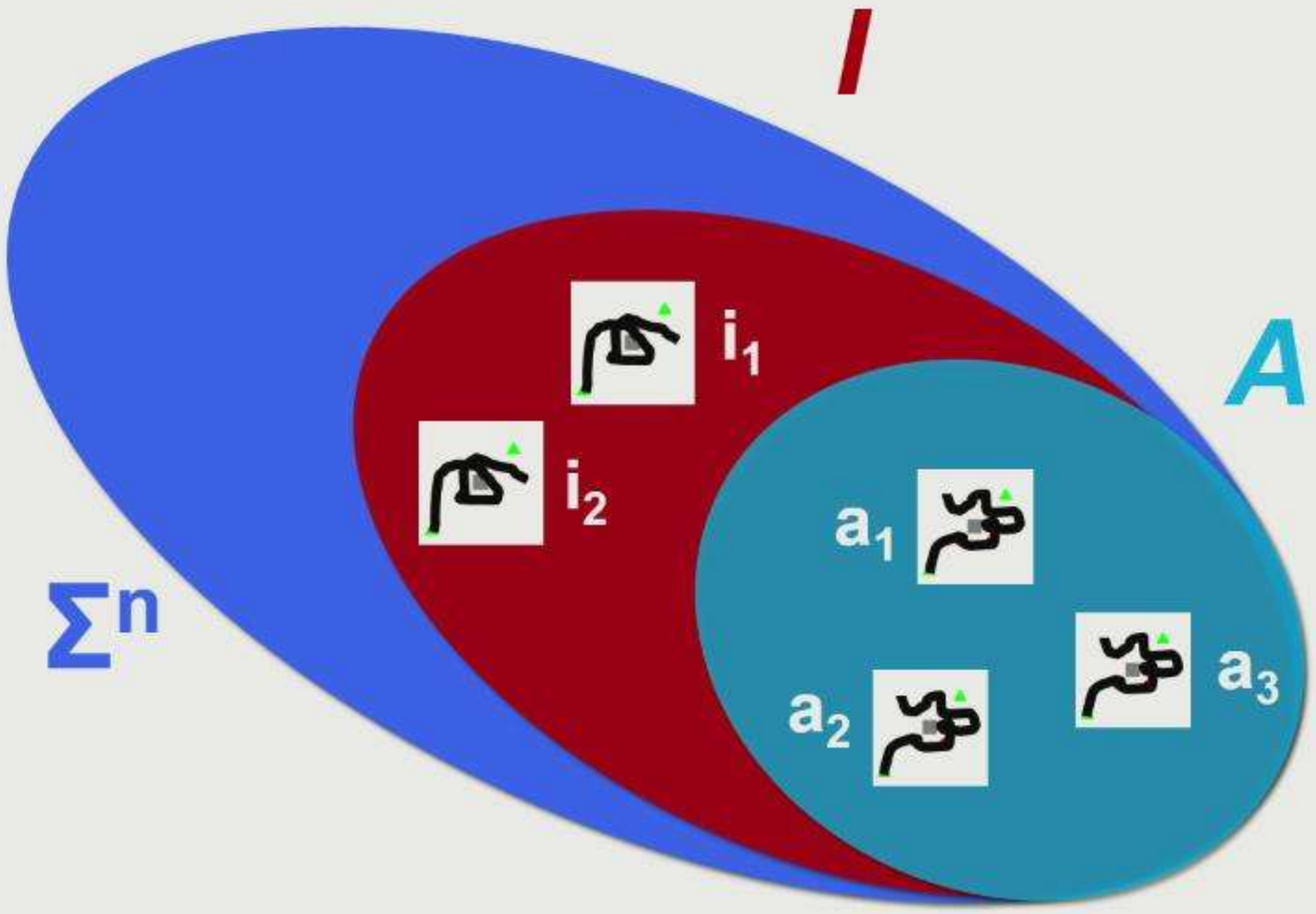
The CI instance $\mathcal{C} = (\mathcal{H}, \mathcal{S}, n, \epsilon, \lambda, \rho)$ is *feasible* if *D* exists.

An *improviser* is a probabilistic algorithm sampling from *D*.

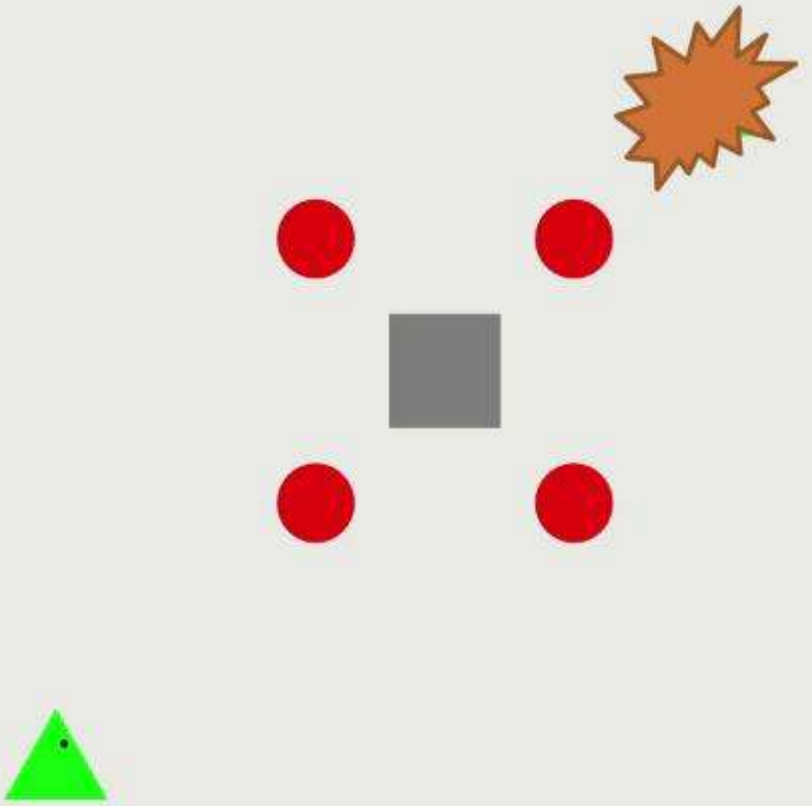
Fremont et al., *Control Improvisation*, arXiv:1704.06319, extending FSTTCS 2015.

Example

- Suppose $A = \{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$ and $I \setminus A = \{\mathbf{i}_1, \mathbf{i}_2\}$
- With $\varepsilon = \rho = 1/4$:
  - Return $\mathbf{a}_1$, $\mathbf{a}_2$, and $\mathbf{a}_3$ with probability ¼
  - Return $\mathbf{i}_1$ and $\mathbf{i}_2$ with probability ⅛



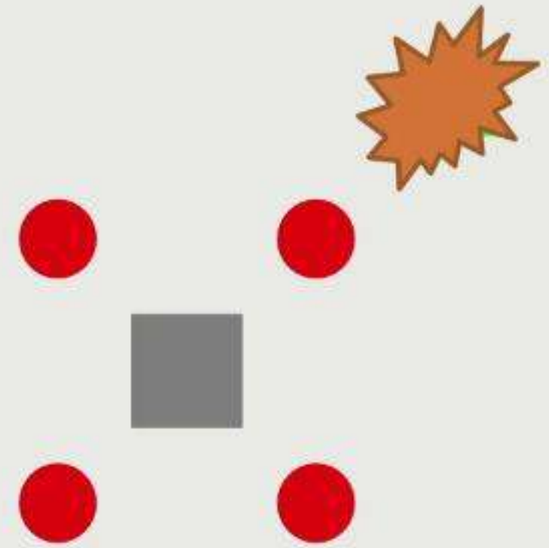
Reactive Control Improvisation



Fremont and Seshia, *Reactive Control Improvisation*, CAV 2018.

Reactive Control Improvisation

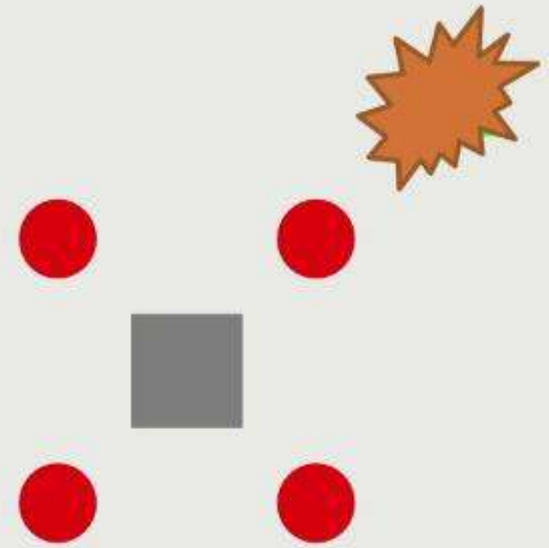
- Changes to definition:
  - System and adversary alternate picking symbols (2-player game)



Fremont and Seshia, *Reactive Control Improvisation*, CAV 2018.

Reactive Control Improvisation

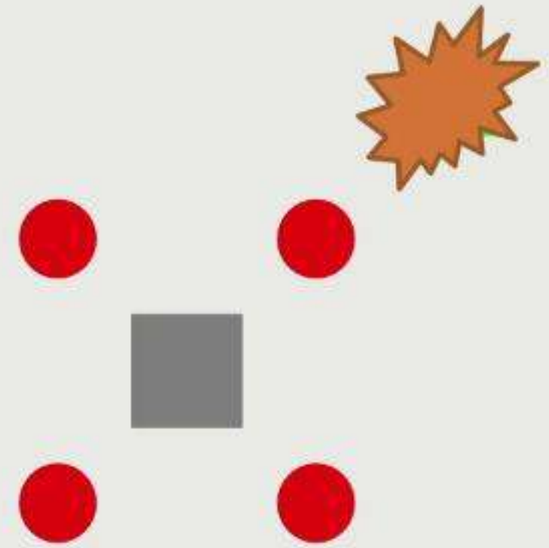
- Changes to definition:
  - System and adversary alternate picking symbols (2-player game)
  - Hard, soft, randomness constraints hold *against every adversary*



Fremont and Seshia, *Reactive Control Improvisation*, CAV 2018.

Reactive Control Improvisation

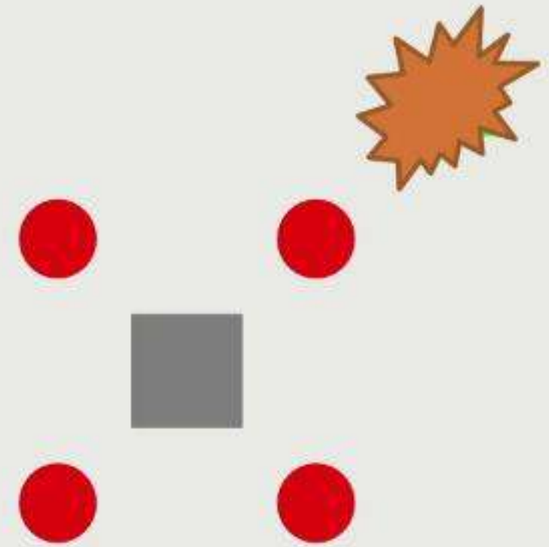
- Changes to definition:
  - System and adversary alternate picking symbols (2-player game)
  - Hard, soft, randomness constraints hold *against every adversary*
  - Improviser → improvising *strategy*



Fremont and Seshia, *Reactive Control Improvisation*, CAV 2018.

Reactive Control Improvisation

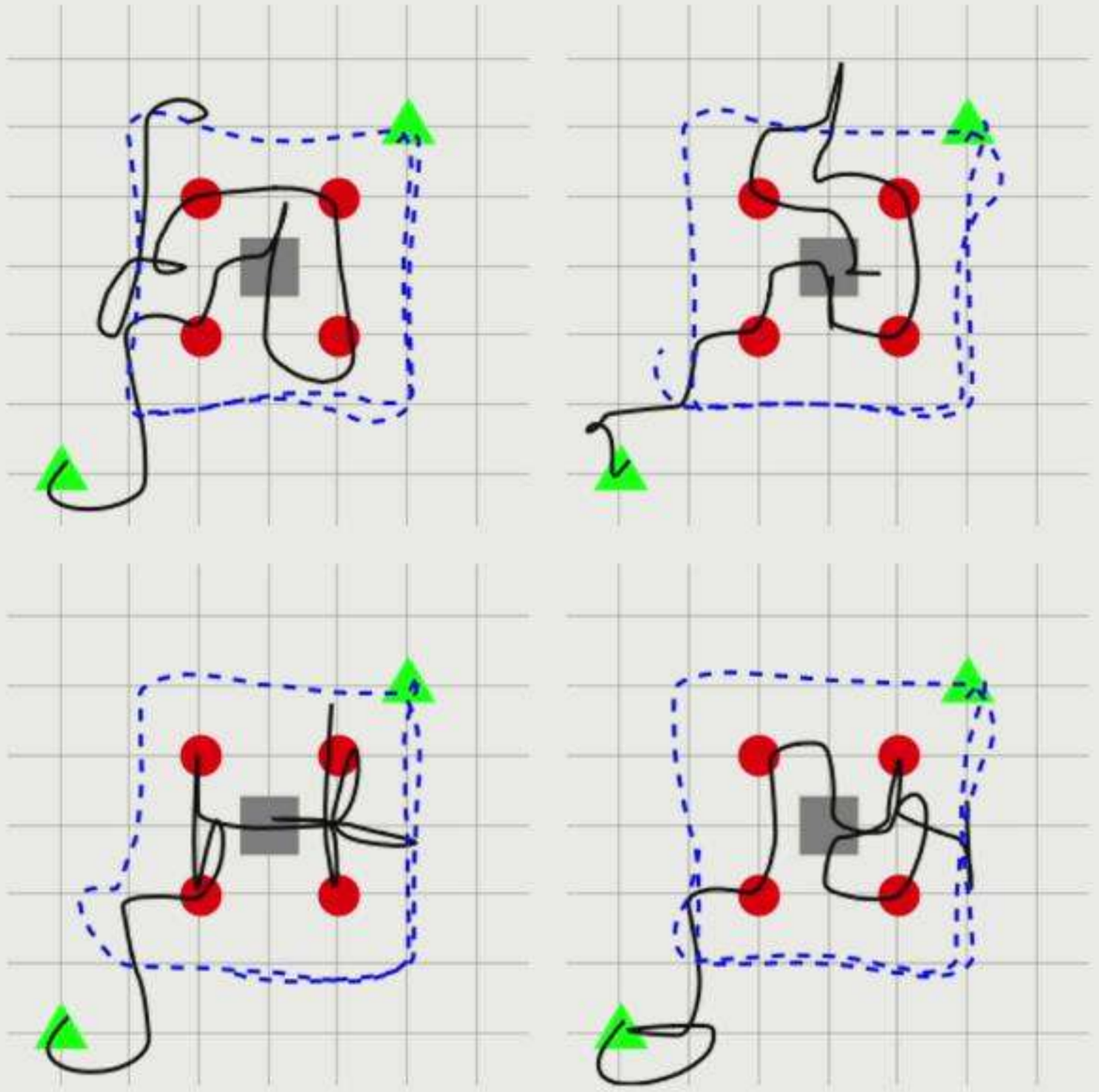
- Changes to definition:
  - System and adversary alternate picking symbols (2-player game)
  - Hard, soft, randomness constraints hold *against every adversary*
  - Improviser → improvising *strategy*
- This enables randomized reactive synthesis (for bounded time)



Fremont and Seshia, *Reactive Control Improvisation*, CAV 2018.

RCI Example

- Hard constraint:
  - Visit the 4 circles in 30 moves, avoiding the adversary
- Soft constraint:
  - 75% of the time, visit each circle exactly once
- Randomness requirement:
  - Use the smallest feasible ρ (about 10^{-12})



Outline

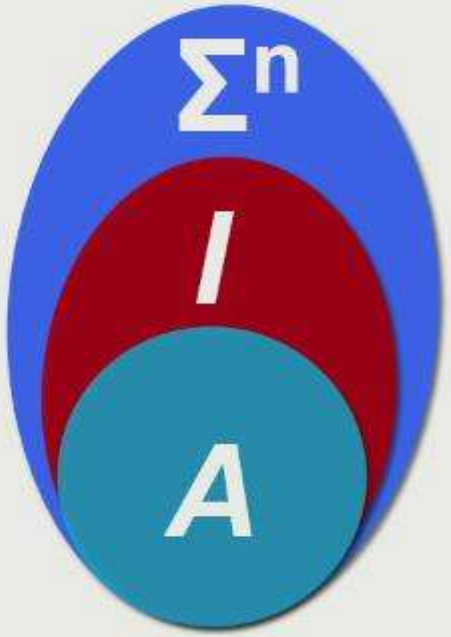
1. Control Improvisation
   - Definition and motivating applications
2. **Theory of CI**
   - Efficient algorithms and hardness results
3. Designing and Analyzing Perception Systems
4. Conclusion & Future Work

Existence of Improvisers

- When is it possible to solve a CI problem?

Existence of Improvisers

- When is it possible to solve a CI problem?
- Feasibility just requires I and A to be large enough (with $\lambda = 0$)

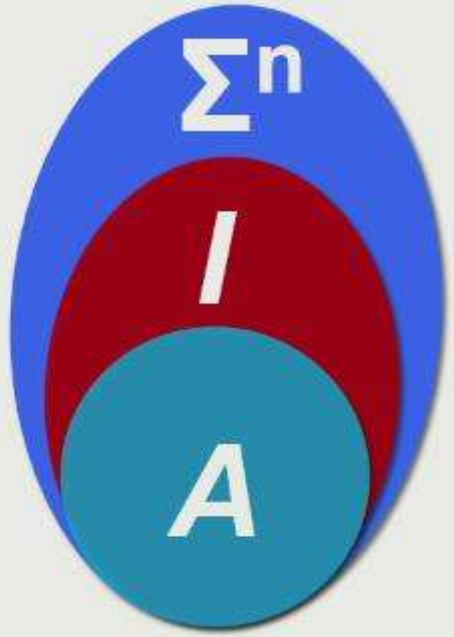


Existence of Improvisers

- When is it possible to solve a CI problem?
- Feasibility just requires *I* and *A* to be large enough (with λ = 0)

Theorem. A CI instance $\mathcal{C} = (H, S, n, \epsilon, 0, \rho)$ is feasible if and only if:

- $|I| \geq 1/\rho$
- $|A| \geq (1 - \epsilon)/\rho$.



Improvisation Schemes

A synthesis algorithm:

Improvisation Schemes

A synthesis algorithm:

A *polynomial-time improvisation scheme* for a class *P* of CI instances is an algorithm S such that given C in *P*,

Improvisation Schemes

A synthesis algorithm:

A *polynomial-time improvisation scheme* for a class *P* of CI instances is an algorithm S such that given C in *P*,

- S returns an improviser if C is feasible, and otherwise ⊥

Improvisation Schemes

A synthesis algorithm:

A *polynomial-time improvisation scheme* for a class *P* of CI instances is an algorithm S such that given C in *P*,

- S returns an improviser if C is feasible, and otherwise ⊥
- S and the improvisers it generates run in time polynomial in |C|

A Generic Improvisation Scheme

A Generic Improvisation Scheme

- If you can efficiently:

A Generic Improvisation Scheme

- If you can efficiently:
  - **Intersect** and take the **difference** of two specifications

A Generic Improvisation Scheme

- If you can efficiently:
  - **Intersect** and take the **difference** of two specifications
  - **Restrict** a specification to a given length

A Generic Improvisation Scheme

- If you can efficiently:
  - **Intersect** and take the **difference** of two specifications
  - **Restrict** a specification to a given length
  - **Count** the number of words satisfying a specification

A Generic Improvisation Scheme

- If you can efficiently:
  - **Intersect** and take the **difference** of two specifications
  - **Restrict** a specification to a given length
  - **Count** the number of words satisfying a specification
  - **Uniformly sample** from those words

A Generic Improvisation Scheme

- If you can efficiently:
  - **Intersect** and take the **difference** of two specifications
  - **Restrict** a specification to a given length
  - **Count** the number of words satisfying a specification
  - **Uniformly sample** from those words

} Nontrivial!
e.g. if spec is a
Boolean formula

A Generic Improvisation Scheme

- If you can efficiently:
  - **Intersect** and take the **difference** of two specifications
  - **Restrict** a specification to a given length
  - **Count** the number of words satisfying a specification
  - **Uniformly sample** from those words

} Nontrivial! e.g. if spec is a Boolean formula

- There is a polynomial-time improvisation scheme:

A Generic Improvisation Scheme

- If you can efficiently:
  - **Intersect** and take the **difference** of two specifications
  - **Restrict** a specification to a given length
  - **Count** the number of words satisfying a specification
  - **Uniformly sample** from those words

} Nontrivial! e.g. if spec is a Boolean formula

- There is a polynomial-time improvisation scheme:
  1. Construct specifications for I, A, and $I \setminus A$

A Generic Improvisation Scheme

- If you can efficiently:
  - **Intersect** and take the **difference** of two specifications
  - **Restrict** a specification to a given length
  - **Count** the number of words satisfying a specification
  - **Uniformly sample** from those words

} Nontrivial! e.g. if spec is a Boolean formula

- There is a polynomial-time improvisation scheme:
  1. Construct specifications for I, A, and $I \setminus A$
  2. Count them and check the feasibility inequalities

A Generic Improvisation Scheme

- If you can efficiently:
  - **Intersect** and take the **difference** of two specifications
  - **Restrict** a specification to a given length
  - **Count** the number of words satisfying a specification
  - **Uniformly sample** from those words

} Nontrivial! e.g. if spec is a Boolean formula

- There is a polynomial-time improvisation scheme:
  1. Construct specifications for I, A, and $I \setminus A$
  2. Count them and check the feasibility inequalities
  3. Sample from A or $I \setminus A$ with the right probabilities

Algorithms and Complexity

Fremont et al., *Control Improvisation*, arXiv:1704.06319, extending FSTTCS 2015.

Algorithms and Complexity

- Poly-time improvisation schemes for:
  - DFAs (examples in music and robotic planning)

Fremont et al., *Control Improvisation*, arXiv:1704.06319, extending FSTTCS 2015.

Algorithms and Complexity

- Poly-time improvisation schemes for:
  - DFAs (examples in music and robotic planning)
  - Unambiguous context-free grammars (fuzz testing)

Fremont et al., *Control Improvisation*, arXiv:1704.06319, extending FSTTCS 2015.

Algorithms and Complexity

- Poly-time improvisation schemes for:
  - DFAs (examples in music and robotic planning)
  - Unambiguous context-free grammars (fuzz testing)

$\mathcal{H}$ \ $\mathcal{S}$		**DFA**	**CFG**		**NFA**
			unamb.	amb.	
DFA		poly-time	poly-time		
CFG	unambiguous	poly-time	#P		
	ambiguous				#P
NFA					

Fremont et al., *Control Improvisation*, arXiv:1704.06319, extending FSTTCS 2015.

Algorithms and Complexity

- Poly-time improvisation schemes for:
  - DFAs (examples in music and robotic planning)
  - Unambiguous context-free grammars (fuzz testing)
- Approximate scheme for Boolean formulas, using SAT solvers

$\mathcal{H}$ \ $\mathcal{S}$		**DFA**	**CFG** unamb.	**CFG** amb.	**NFA**
DFA		poly-time	poly-time		
CFG	unambiguous	poly-time	#P		
CFG	ambiguous				#P
NFA					

Fremont et al., *Control Improvisation*, arXiv:1704.06319, extending FSTTCS 2015.

Feasibility of Reactive Control Improvisation

Feasibility of Reactive Control Improvisation

- Feasibility (realizability) again reduces to counting: there must be *sufficiently many* ways to win the 2-player game

Feasibility of Reactive Control Improvisation

- Feasibility (realizability) again reduces to counting: there must be *sufficiently many* ways to win the 2-player game

Theorem. An RCI instance is feasible if and only if:

- $W(I) \geq 1/\rho$
- $W(A) \geq (1-\epsilon)/\rho$.

Width

- Number of “winning” plays, minimized over all adversaries

Width

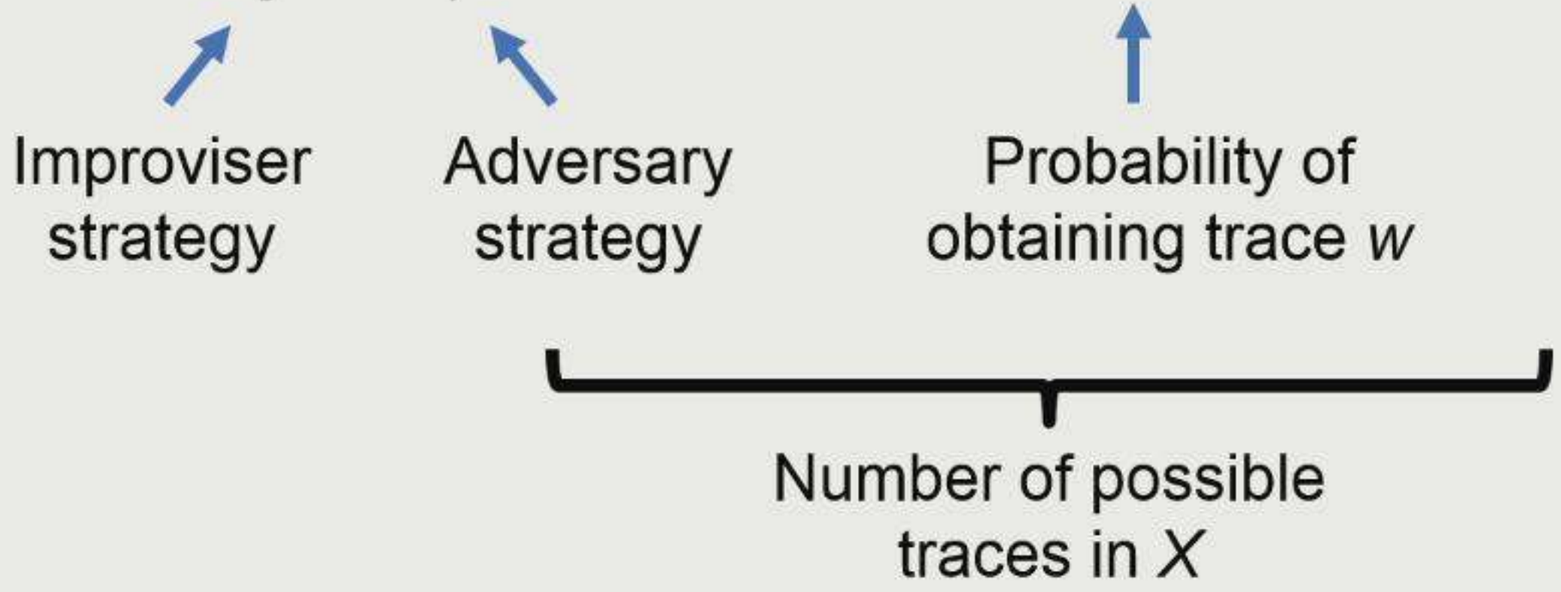
- Number of “winning” plays, minimized over all adversaries

$$W(X) = \max_{\sigma} \min_{\tau} |\{w \in X \mid P_{\sigma,\tau}(w) > 0\}|$$

Width

- Number of “winning” plays, minimized over all adversaries

$$W(X) = \max_{\sigma} \min_{\tau} |\{w \in X \mid P_{\sigma,\tau}(w) > 0\}|$$

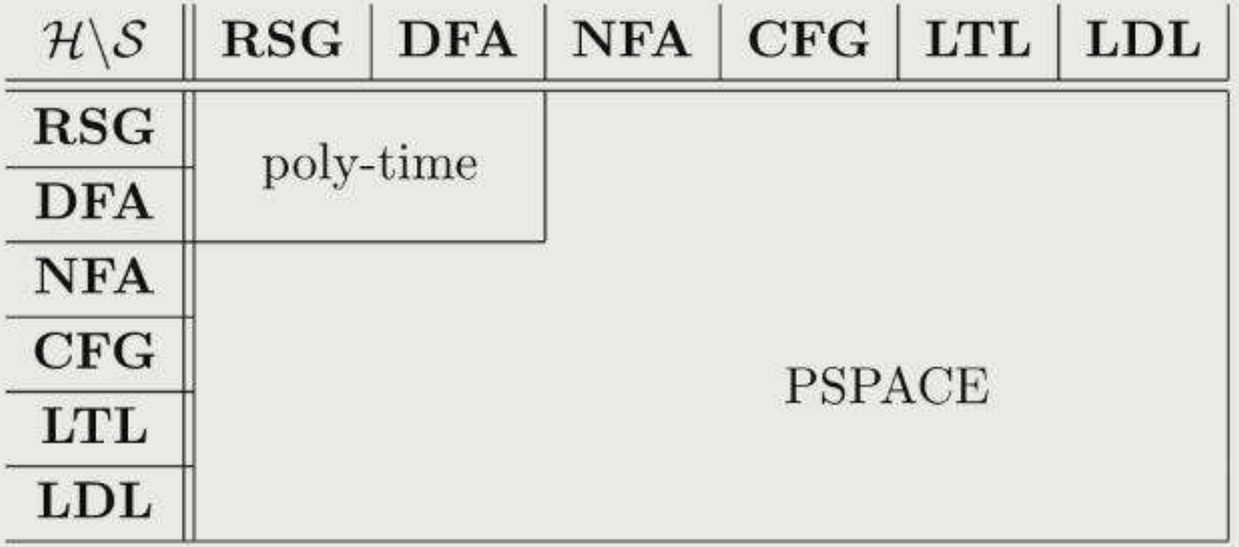


Algorithms for RCI

Fremont and Seshia, *Reactive Control Improvisation*, CAV 2018.

Algorithms for RCI

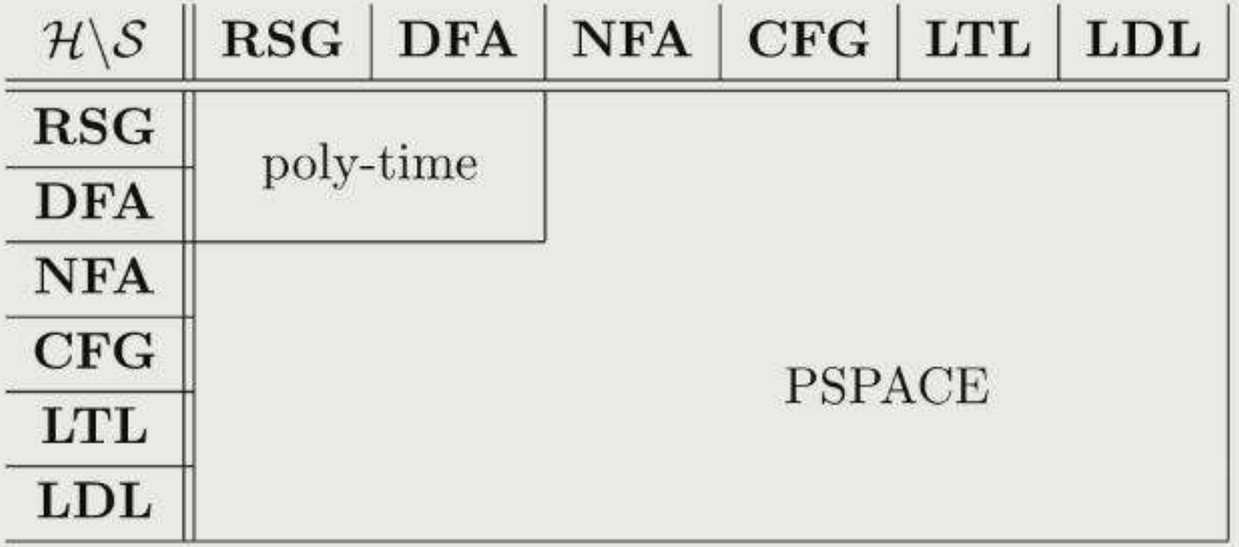
- Again, efficient scheme for DFA specifications



Fremont and Seshia, *Reactive Control Improvisation*, CAV 2018.

Algorithms for RCI

- Again, efficient scheme for DFA specifications
- PSPACE-equivalent for temporal logics



Fremont and Seshia, *Reactive Control Improvisation*, CAV 2018.

Outline

1. Control Improvisation
   - Definition and motivating applications
2. Theory of CI
   - Efficient algorithms and hardness results
3. **Designing and Analyzing Perception Systems**
4. Conclusion & Future Work

Motivating Example: an Autonomous Vehicle

Motivating Example: an Autonomous Vehicle

- At any moment in time, the *scene* is the environment configuration (object positions, colors, etc.)

Motivating Example: an Autonomous Vehicle

- At any moment in time, the *scene* is the environment configuration (object positions, colors, etc.)
- Collecting examples of real-world scenes is expensive

Motivating Example: an Autonomous Vehicle

- At any moment in time, the *scene* is the environment configuration (object positions, colors, etc.)
- Collecting examples of real-world scenes is expensive
- We would like to generate realistic scenes automatically for training or testing

Generating Meaningful Scenes

- Scene space is large, high-dimensional


Car Model


Car Location


Car Orientation


Number of Cars


Reference


Scene Background


Car Color


Weather


Time of Day

Generating Meaningful Scenes

- Scene space is large, high-dimensional
- Realistic scenes have complex geometric constraints

Generating Meaningful Scenes

- Scene space is large, high-dimensional
- Realistic scenes have complex geometric constraints
- We may be interested in particular types of scenes (e.g. highway traffic)


Car Model


Car Location


Car Orientation


Number of Cars


Reference


Scene Background


Car Color


Weather


Time of Day

Generating Meaningful Scenes

- Scene space is large, high-dimensional
- Realistic scenes have complex geometric constraints
- We may be interested in particular types of scenes (e.g. highway traffic)
- Generating these is another instance of CI, but with what kind of specification?


Car Model


Car Location


Car Orientation


Number of Cars


Reference


Scene Background


Car Color


Weather


Time of Day

Specifications for Scenes

- Again, hard, soft, randomness constraints:

- Objects should not intersect
- 20% of the images should be at night

Specifications for Scenes

- Again, hard, soft, randomness constraints:

- Objects should not intersect
- 20% of the images should be at night
- The gap between cars should follow this distribution…

Specifications for Scenes

- Again, hard, soft, randomness constraints:

- Objects should not intersect
- 20% of the images should be at night
- The gap between cars should follow this distribution…

- How to encode these? A *probabilistic programming language*

SCENIC: a Scenario Description Language

- Defines a *distribution over scenes*

Fremont et al., *Scenic: A Language for Scenario Specification and Scene Generation*, PLDI 2019.

SCENIC: a Scenario Description Language

- Defines a *distribution over scenes*

Bumper-to-Bumper Traffic
(~20 lines of Scenic)

Fremont et al., *Scenic: A Language for Scenario Specification and Scene Generation*, PLDI 2019.

SCENIC: a Scenario Description Language

- Defines a *distribution over scenes*

Bumper-to-Bumper Traffic
(~20 lines of Scenic)

Fremont et al., *Scenic: A Language for Scenario Specification and Scene Generation*, PLDI 2019.

SCENIC: a Scenario Description Language

- Defines a *distribution over scenes*
- Readable, concise syntax for common geometric relationships

Bumper-to-Bumper Traffic
(~20 lines of Scenic)

Fremont et al., *Scenic: A Language for Scenario Specification and Scene Generation*, PLDI 2019.

SCENIC: a Scenario Description Language

- Defines a *distribution over scenes*
- Readable, concise syntax for common geometric relationships
- Domain-specific sampling techniques based on configuration spaces

Bumper-to-Bumper Traffic
(~20 lines of Scenic)

Fremont et al., *Scenic: A Language for Scenario Specification and Scene Generation*, PLDI 2019.

SCENIC: a Scenario Description Language

- Defines a *distribution over scenes*
- Readable, concise syntax for common geometric relationships
- Domain-specific sampling techniques based on configuration spaces

Bumper-to-Bumper Traffic
(~20 lines of Scenic)

Fremont et al., *Scenic: A Language for Scenario Specification and Scene Generation*, PLDI 2019.

SCENIC: a Scenario Description Language

- Defines a *distribution over scenes*
- Readable, concise syntax for common geometric relationships
- Domain-specific sampling techniques based on configuration spaces

Bumper-to-Bumper Traffic
(~20 lines of Scenic)

Fremont et al., *Scenic: A Language for Scenario Specification and Scene Generation*, PLDI 2019.

SCENIC: a Scenario Description Language

- Defines a *distribution over scenes*
- Readable, concise syntax for common geometric relationships
- Domain-specific sampling techniques based on configuration spaces
- Declarative hard and soft constraints

Bumper-to-Bumper Traffic
(~20 lines of Scenic)

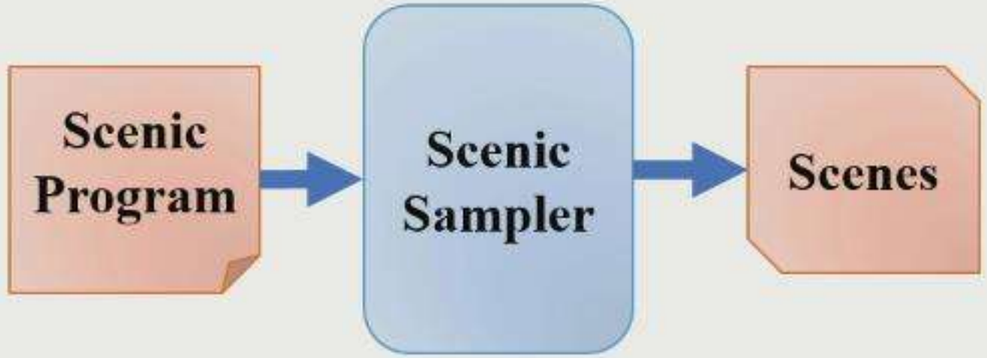
Fremont et al., *Scenic: A Language for Scenario Specification and Scene Generation*, PLDI 2019.

Designing and Analyzing Systems using a PPL

parkedCar.sc

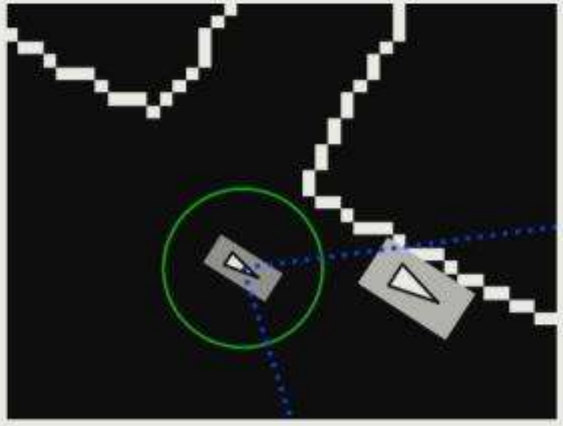
```
ego = Car
Car offset by ...,
    facing ...
```

Designing and Analyzing Systems using a PPL

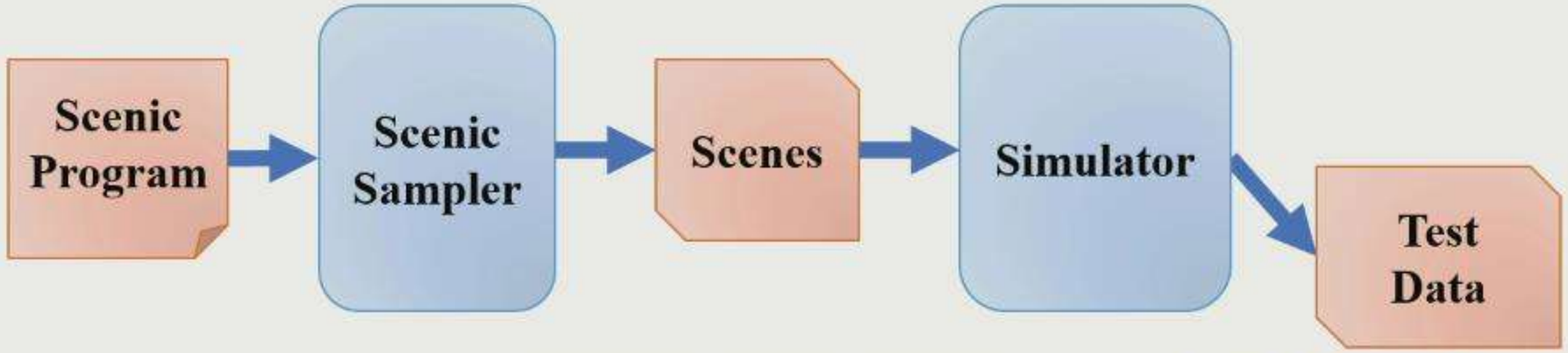


parkedCar.sc

```
ego = Car
Car offset by ...,
    facing ...
```

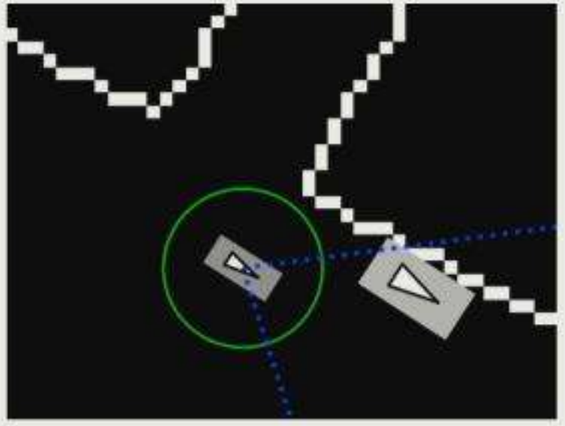


Designing and Analyzing Systems using a PPL

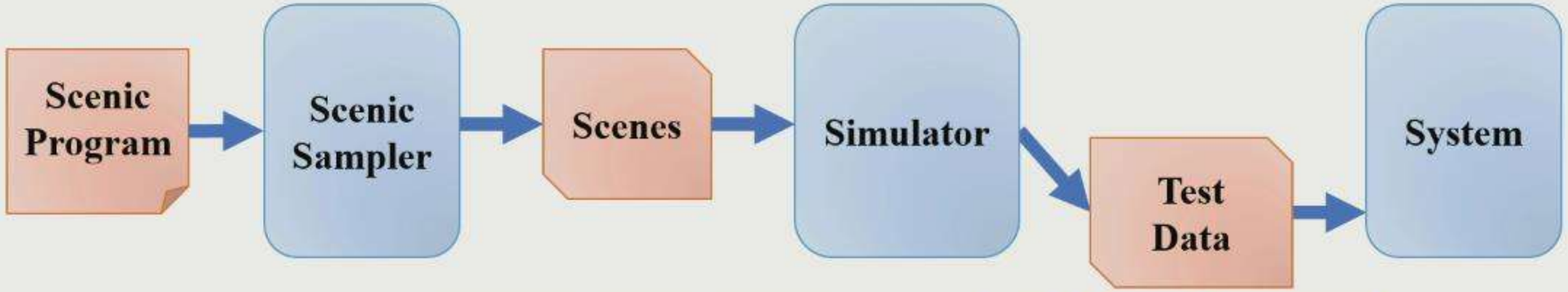


parkedCar.sc

```
ego = Car
Car offset by ...,
    facing ...
```

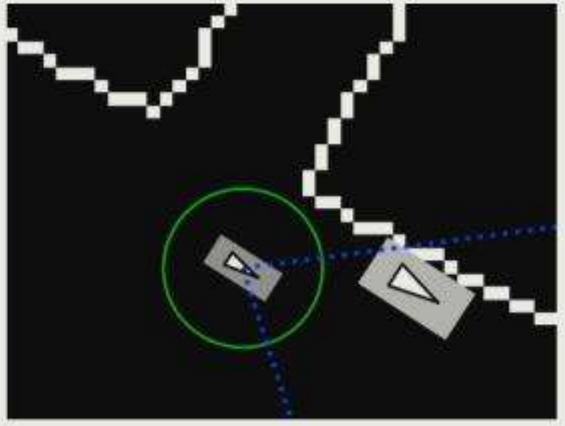


Designing and Analyzing Systems using a PPL

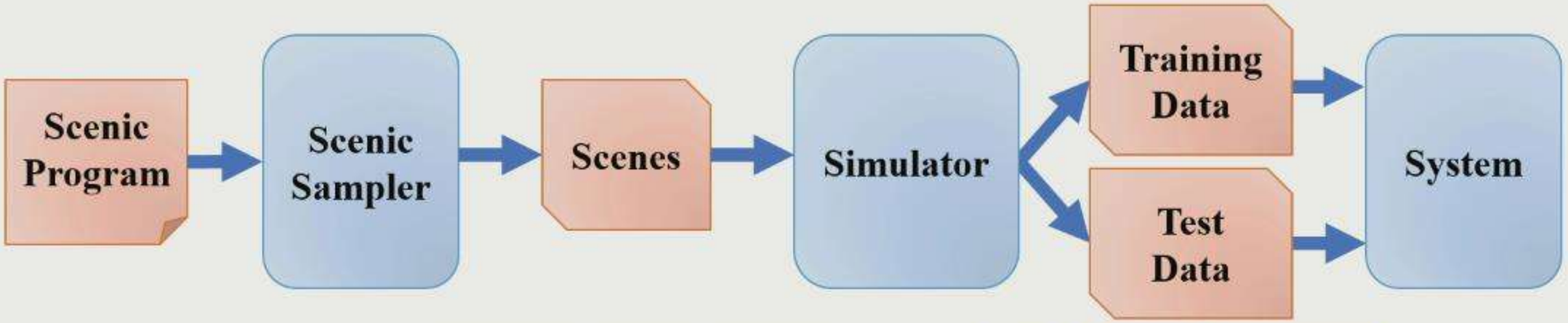


parkedCar.sc

```
ego = Car
Car offset by ...,
    facing ...
```

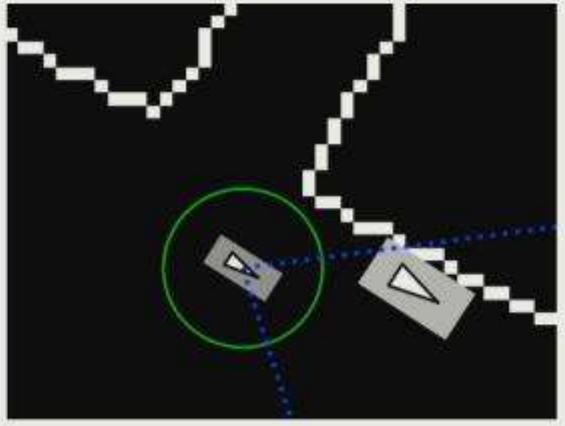


Designing and Analyzing Systems using a PPL

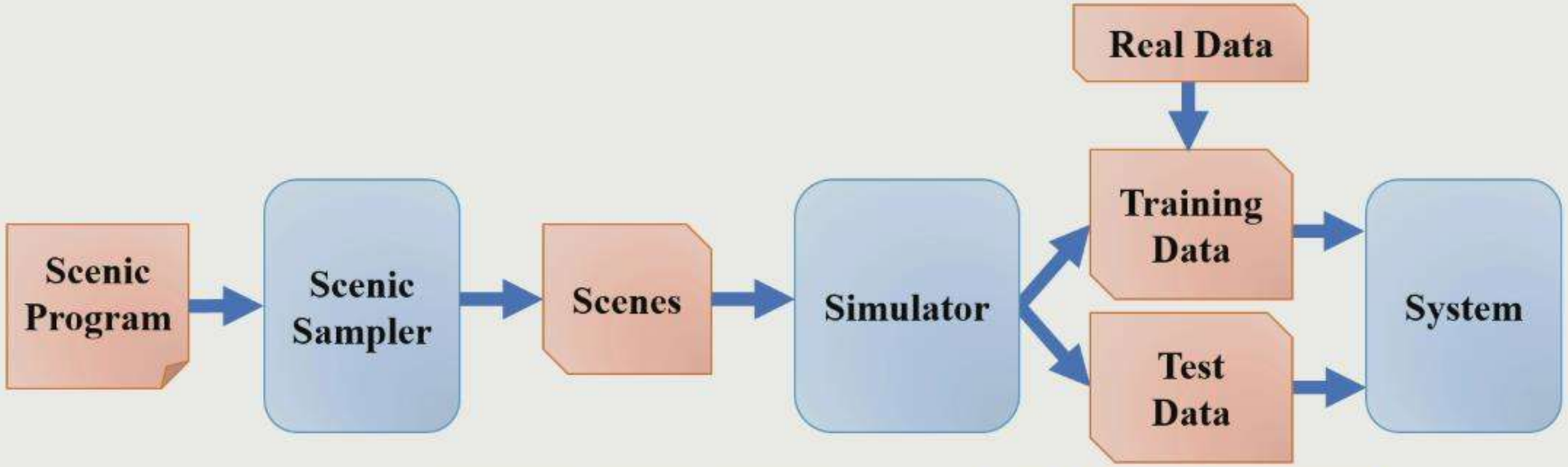


parkedCar.sc

```
ego = Car
Car offset by ...,
    facing ...
```

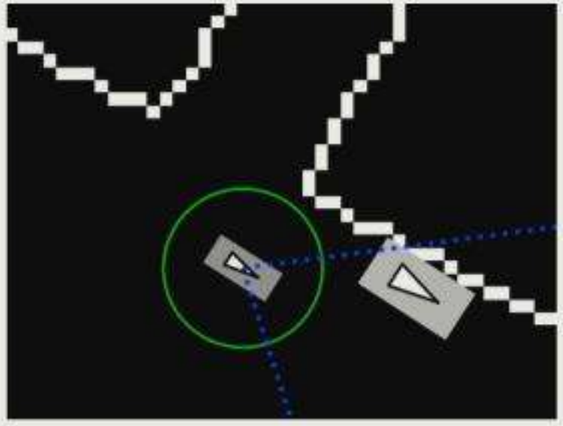


Designing and Analyzing Systems using a PPL

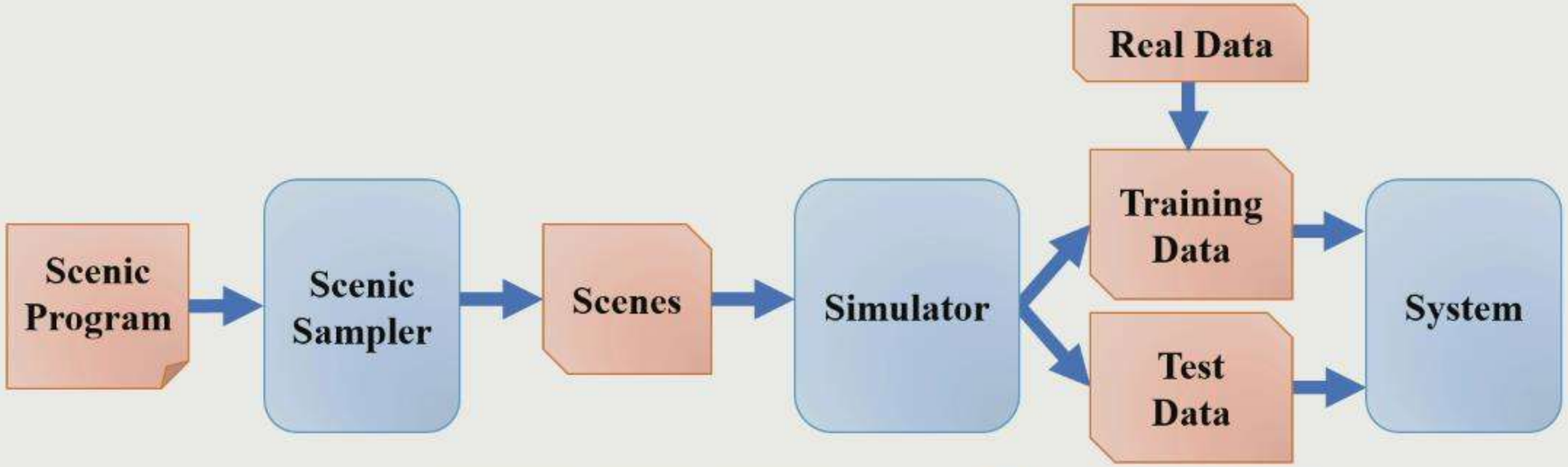


parkedCar.sc

```
ego = Car
Car offset by ...,
    facing ...
```

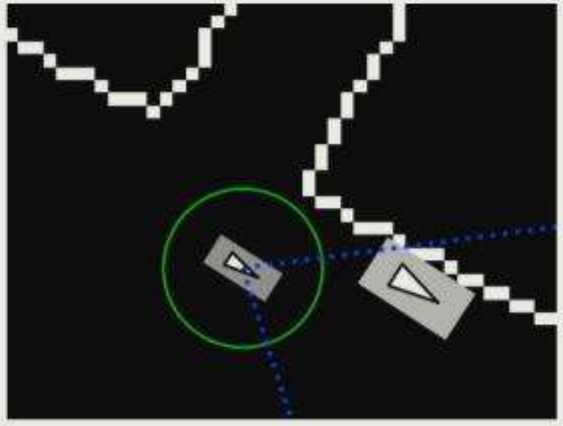


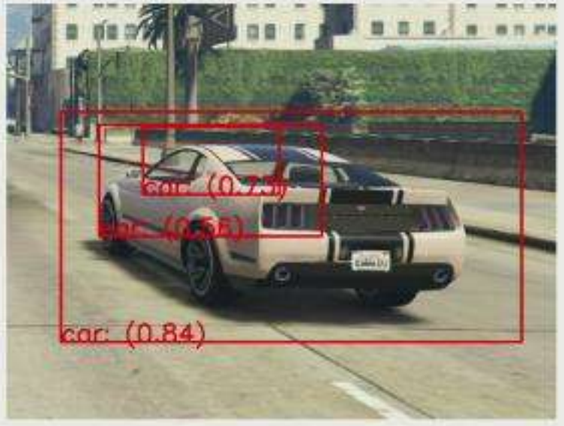
Designing and Analyzing Systems using a PPL



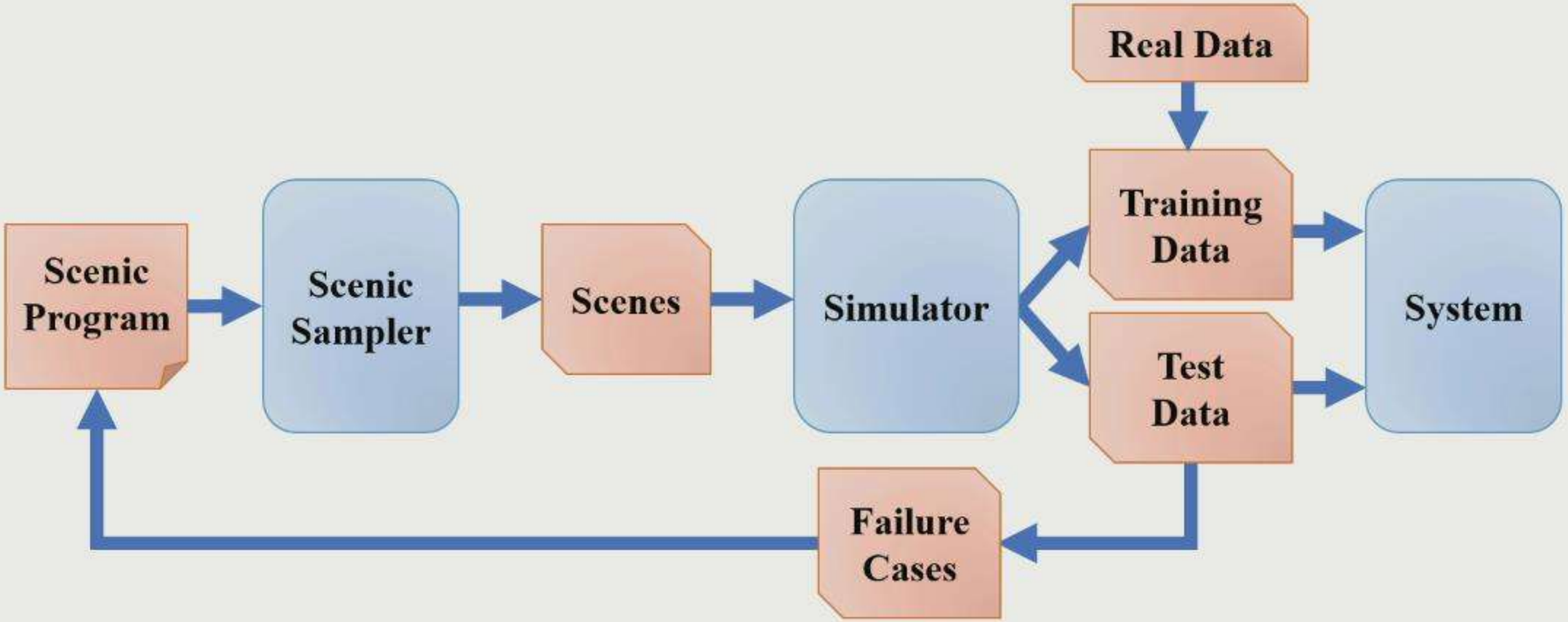
parkedCar.sc

```
ego = Car
Car offset by ...,
    facing ...
```





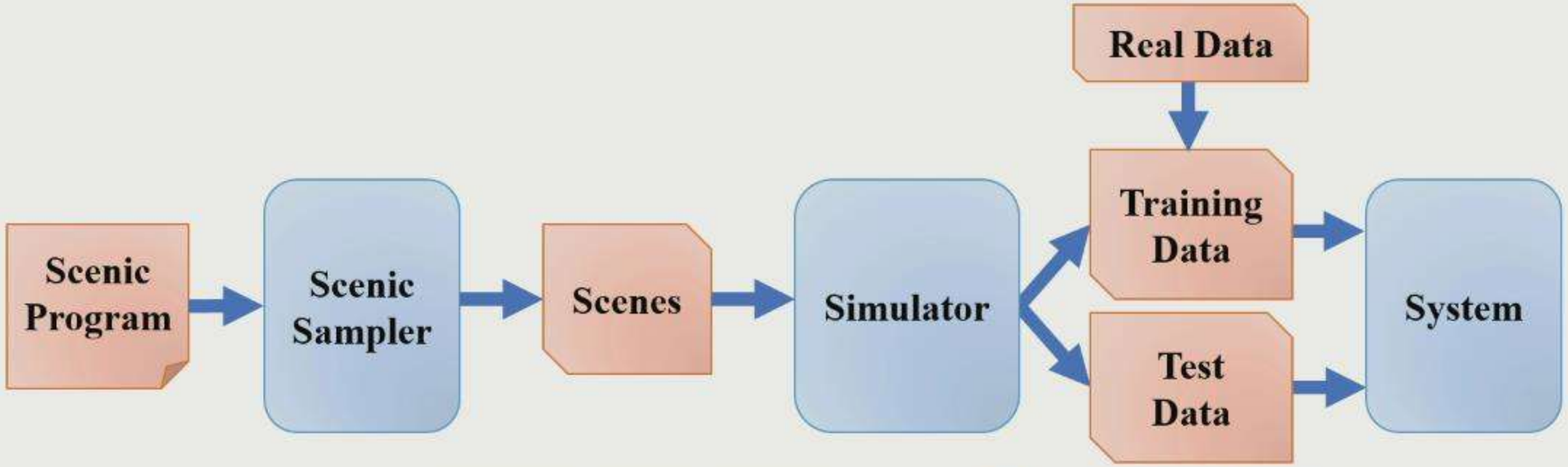
Designing and Analyzing Systems using a PPL



Related Work: PPLs

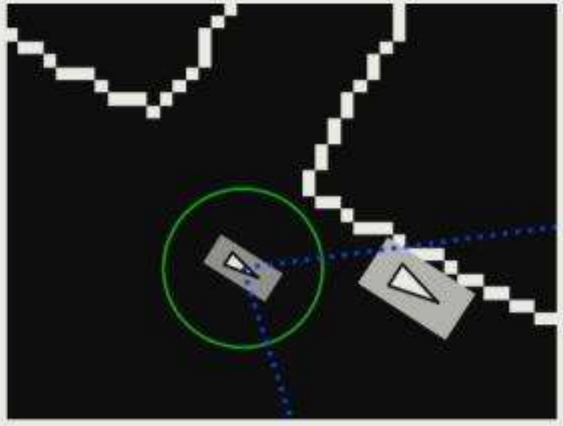
[1] Gordon et al., *FOSE 2014*.
[2] Ritchie, *NIPS Workshop on Prob. Prog.*, 2014.
[3] Kulkarni et al., *CVPR 2015*.

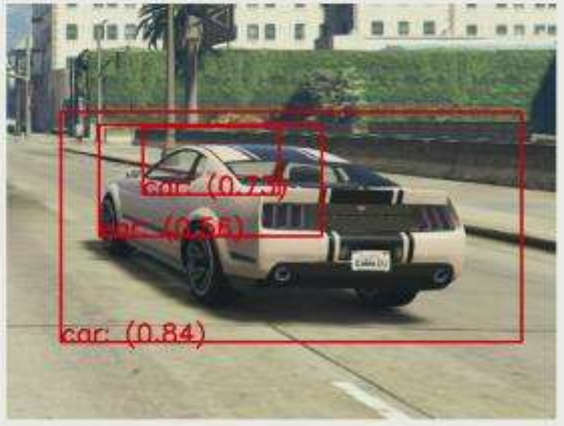
Designing and Analyzing Systems using a PPL



parkedCar.sc

```
ego = Car
Car offset by ...,
    facing ...
```





Related Work: PPLs

- SCENIC's semantics is that of a standard imperative PPL (e.g. [1])
- PPLs have been used as generative models of graphics
  - e.g. Quicksand [2], Picture [3]

[1] Gordon et al., *FOSE 2014*.
[2] Ritchie, *NIPS Workshop on Prob. Prog.*, 2014.
[3] Kulkarni et al., *CVPR 2015*.

Related Work: PPLs

- SCENIC's semantics is that of a standard imperative PPL (e.g. [1])
- PPLs have been used as generative models of graphics
  - e.g. Quicksand [2], Picture [3]
- The main differences in SCENIC:
  - emphasis on generation, not inference
  - domain-specificity

[1] Gordon et al., *FOSE 2014*.
[2] Ritchie, *NIPS Workshop on Prob. Prog.*, 2014.
[3] Kulkarni et al., *CVPR 2015*.

Related Work: Graphics & ML

[4] Fisher et al., *SIGGRAPH 2012.*
[5] Jiang et al., *Int. J. Computer Vision*, 2018.
[6] Johnson-Roberson et al., *ICRA 2017.*
[7] Wong et al., *DICTA 2016.*

Related Work: Graphics & ML

- Scene synthesis from examples or grammars [4, 5]
- Data augmentation/generation in ML [6, 7]

[4] Fisher et al., *SIGGRAPH 2012.*
[5] Jiang et al., *Int. J. Computer Vision*, 2018.
[6] Johnson-Roberson et al., *ICRA 2017.*
[7] Wong et al., *DICTA 2016.*

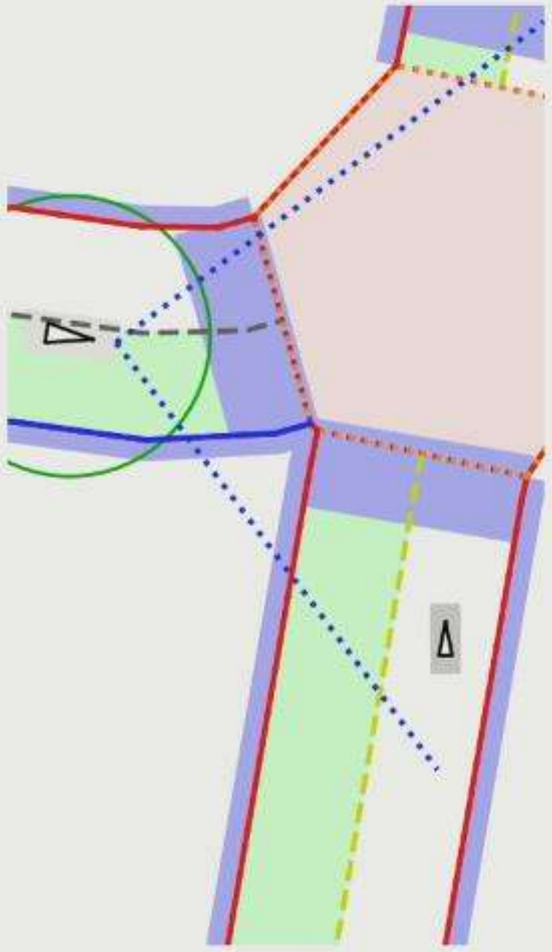
Related Work: Graphics & ML

- Scene synthesis from examples or grammars [4, 5]
- Data augmentation/generation in ML [6, 7]
- The main differences in SCENIC:
  - more control
  - easy to use & interpret

[4] Fisher et al., *SIGGRAPH 2012*.
[5] Jiang et al., *Int. J. Computer Vision*, 2018.
[6] Johnson-Roberson et al., *ICRA 2017*.
[7] Wong et al., *DICTA 2016*.

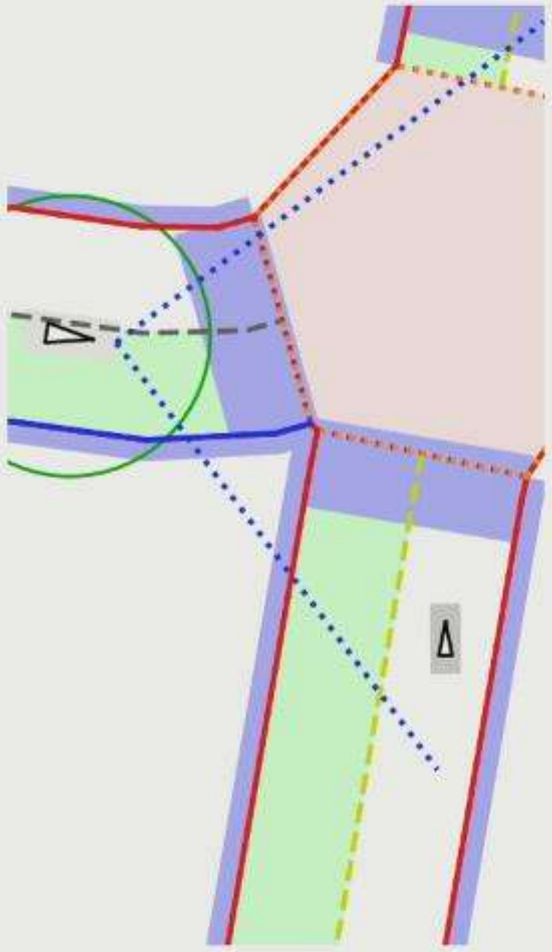
Example: a Badly-Parked Car

Example: a Badly-Parked Car



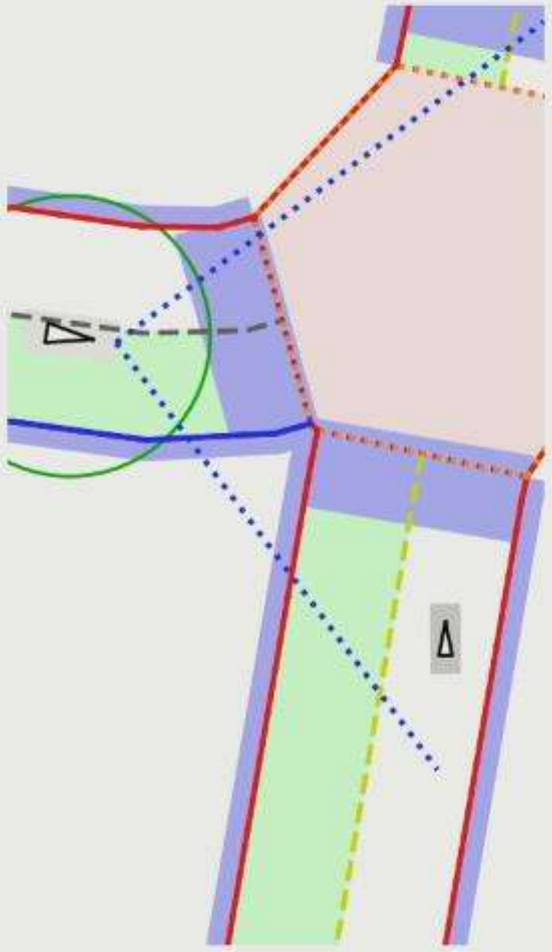
```
from gta import Car, curb, roadDirection
```

Example: a Badly-Parked Car



```
from gta import Car, curb, roadDirection

ego = Car
```

Example: a Badly-Parked Car



```
from gta import Car, curb, roadDirection

ego = Car

spot = OrientedPoint on visible curb
```

Example: a Badly-Parked Car

```
from gta import Car, curb, roadDirection

ego = Car

spot = OrientedPoint on visible curb
```

Example: a Badly-Parked Car

```
from gta import Car, curb, roadDirection

ego = Car

spot = OrientedPoint on visible curb
```

Example: a Badly-Parked Car

```
from gta import Car, curb, roadDirection

ego = Car

spot = OrientedPoint on visible curb
```

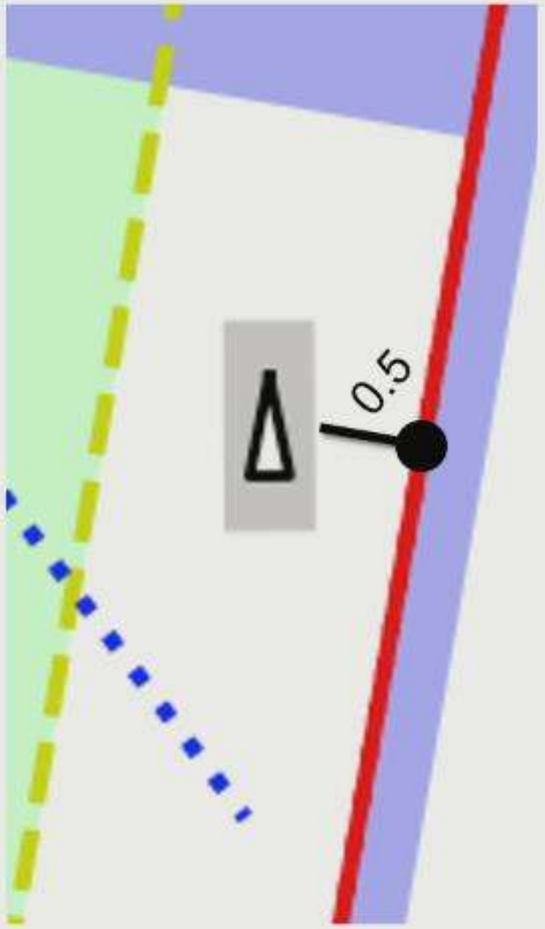
Example: a Badly-Parked Car

```
from gta import Car, curb, roadDirection

ego = Car

spot = OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) * (10, 20) deg
```

Example: a Badly-Parked Car

```
from gta import Car, curb, roadDirection

ego = Car

spot = OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) * (10, 20) deg
Car left of (spot offset by -0.5 @ 0),
```

Example: a Badly-Parked Car




```
from gta import Car, curb, roadDirection

ego = Car

spot = OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) * (10, 20) deg
Car left of (spot offset by -0.5 @ 0),
    facing badAngle relative to roadDirection
```

Example: a Badly-Parked Car

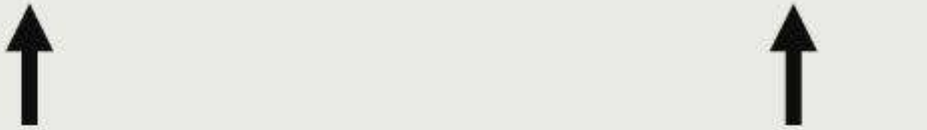
Example: a Badly-Parked Car

Example: a Badly-Parked Car

Specifiers

- Syntactic elements defining properties of an object
  - Can be combined in arbitrary ways to build up a complete definition

```
Car at 120 @ 70, facing toward carWash
```



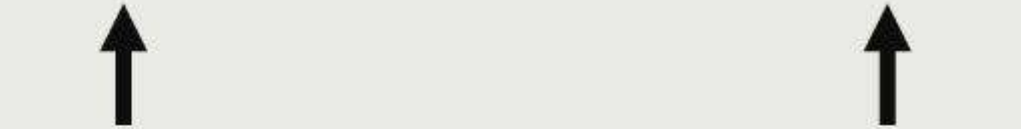
Specifies
position

Specifies
heading

Specifiers

- Syntactic elements defining properties of an object
  - Can be combined in arbitrary ways to build up a complete definition

```
Car at 120 @ 70, facing toward carWash
```



Specifies *position*

Specifies *heading*

- SCENIC has 9 completely different *position* specifiers
  - All mirroring ways we use natural language to talk about positions

Specifiers Capture Dependencies

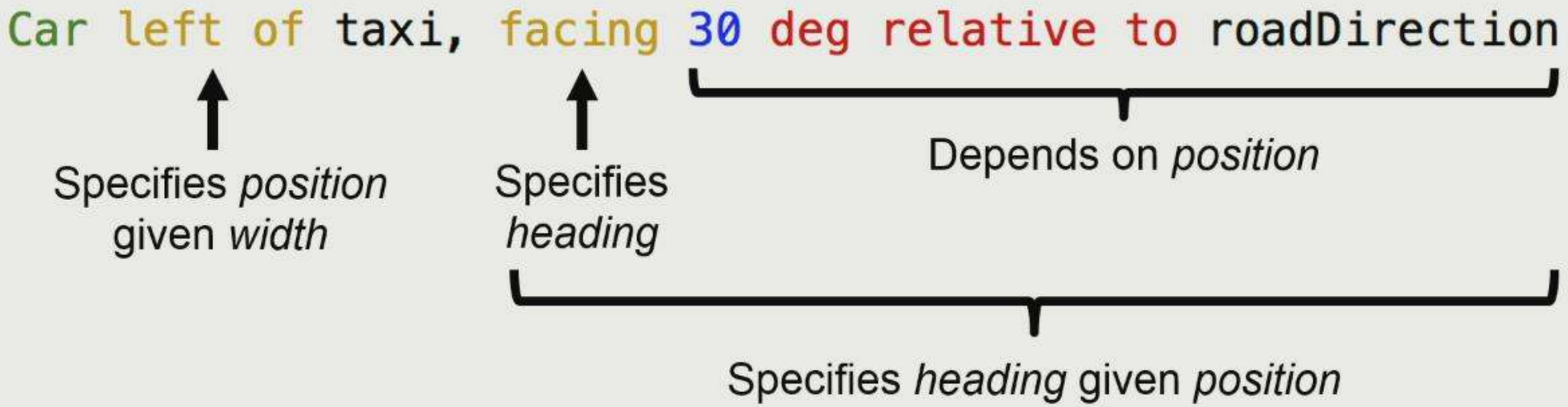
- Can depend on other properties and other specifiers

```
Car left of taxi, facing 30 deg relative to roadDirection
```

Specifies *position*
given *width*

Specifiers Capture Dependencies

- Can depend on other properties and other specifiers



Domain-Specific Sampling Techniques

- Prune infeasible parts of the space by dilating polygons

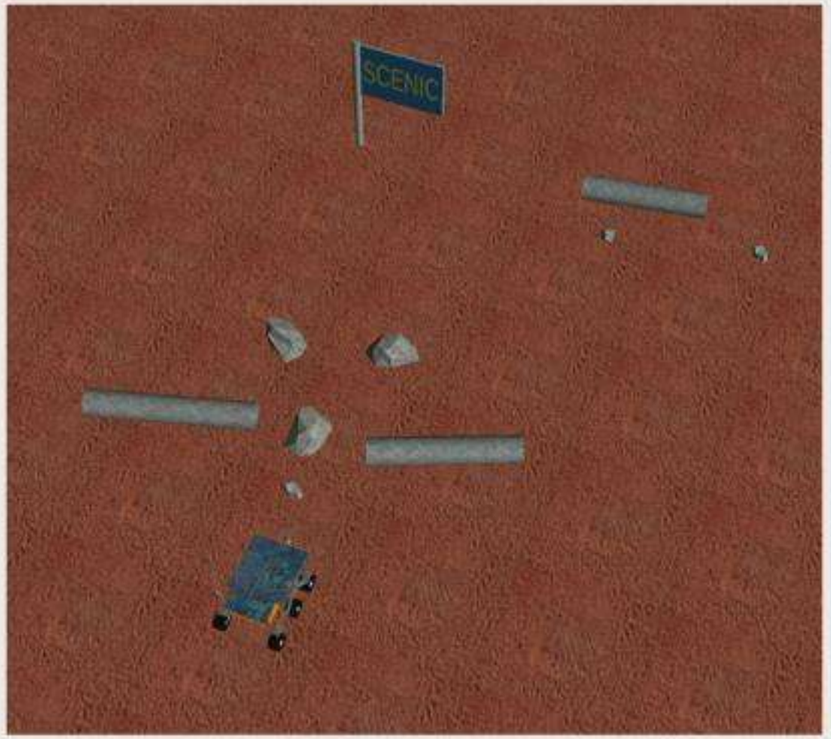
Domain-Specific Sampling Techniques

- Prune infeasible parts of the space by dilating polygons

```
require distance to taxi <= 5
require 15 deg <= (relative
  heading of taxi) <= 45 deg
```

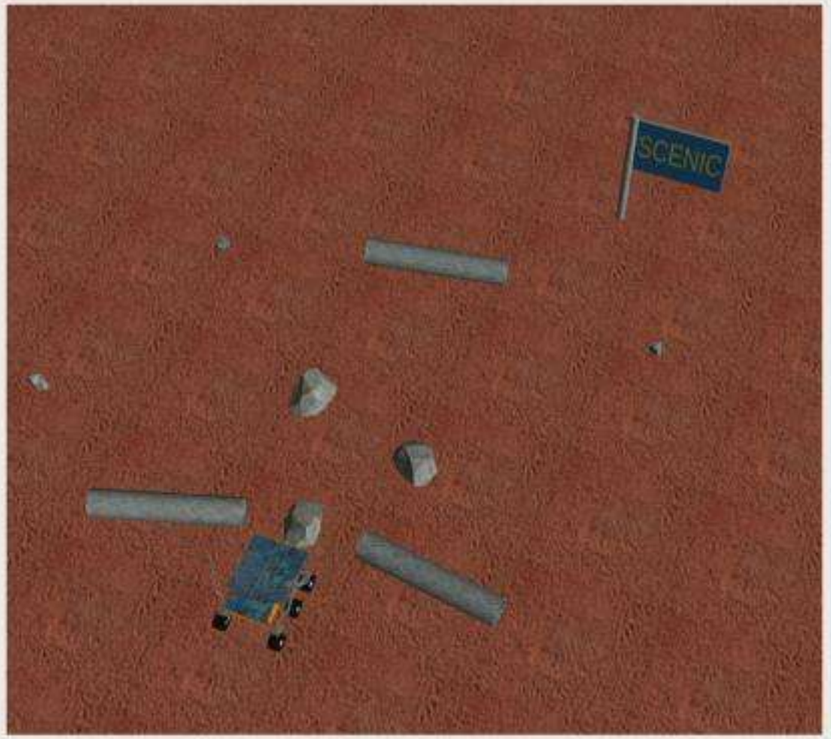
Applications of Scenic

- Exploring system performance
  - Generating specialized test sets



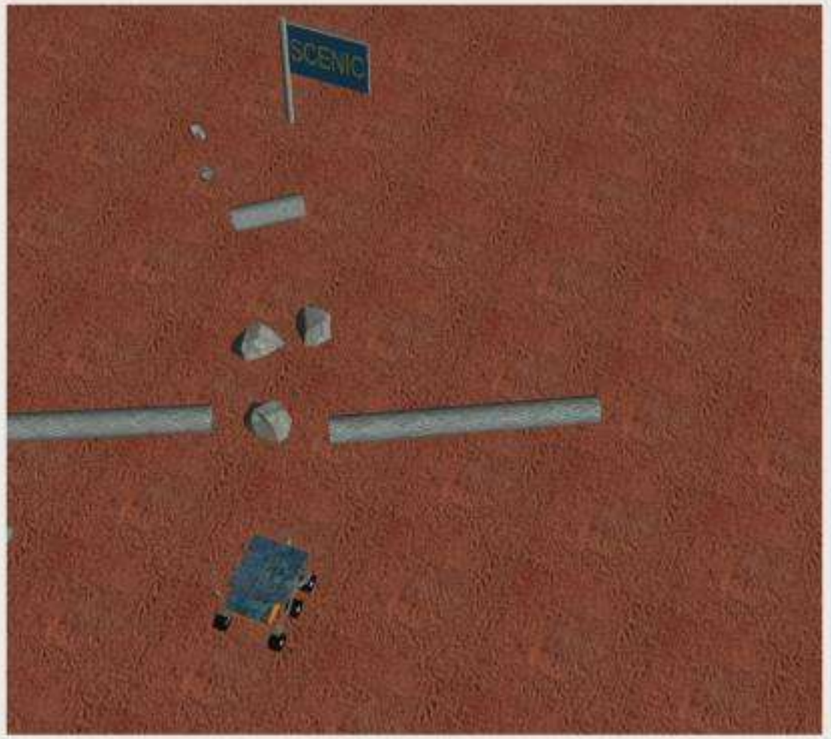
Applications of Scenic

- Exploring system performance
  - Generating specialized test sets
- Debugging a known failure
  - Generalizing in different directions



Applications of Scenic

- Exploring system performance
  - Generating specialized test sets
- Debugging a known failure
  - Generalizing in different directions
- Designing more effective training sets
  - Training on hard cases



Outline

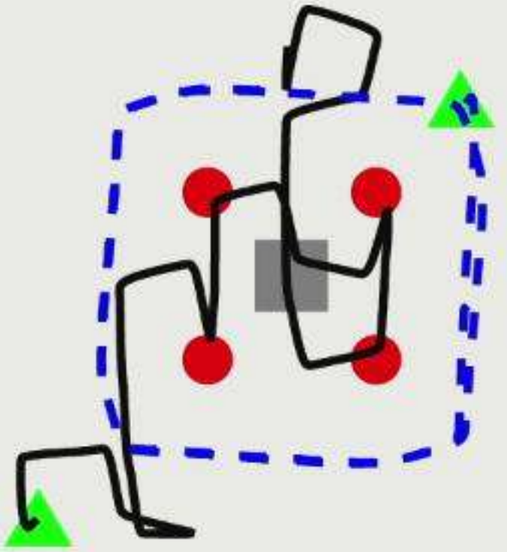
1. Control Improvisation
   - Definition and motivating applications
2. Theory of CI
   - Efficient algorithms and hardness results
3. Designing and Analyzing Perception Systems
4. **Conclusion and Future Work**

Summary and Conclusions

- Algorithmic Improvisation: a framework for randomized synthesis
  - Data generation, planning, fuzz testing, music improvisation, human modeling…

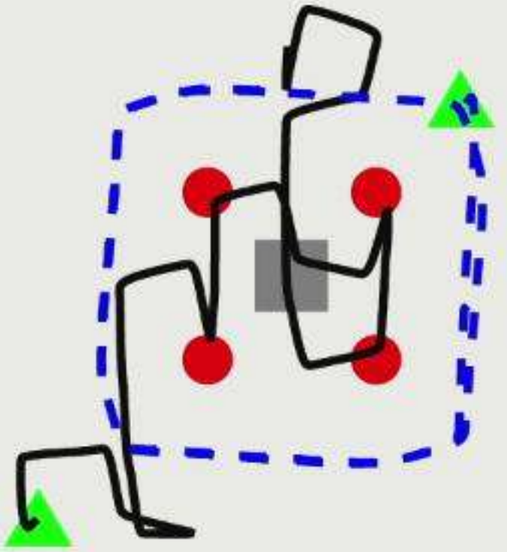
Summary and Conclusions

- Algorithmic Improvisation: a framework for randomized synthesis
  - Data generation, planning, fuzz testing, music improvisation, human modeling…
- Applications to safe autonomy
  - Controller synthesis, synthetic data generation



Summary and Conclusions

- Algorithmic Improvisation: a framework for randomized synthesis
  - Data generation, planning, fuzz testing, music improvisation, human modeling…
- Applications to safe autonomy
  - Controller synthesis, synthetic data generation
- Randomized formal methods can contribute in many ways to the design and verification of autonomous systems



The Future of Algorithmic Improvisation

- Fundamentally new problem in CS → rich possibilities at all levels

The Future of Algorithmic Improvisation

- Fundamentally new problem in CS → rich possibilities at all levels
- Intelligent data generation for a variety of data types
- Environment modeling, including modeling dynamic agents
- Generalized theory of CI enabling new applications

The Future of Algorithmic Improvisation

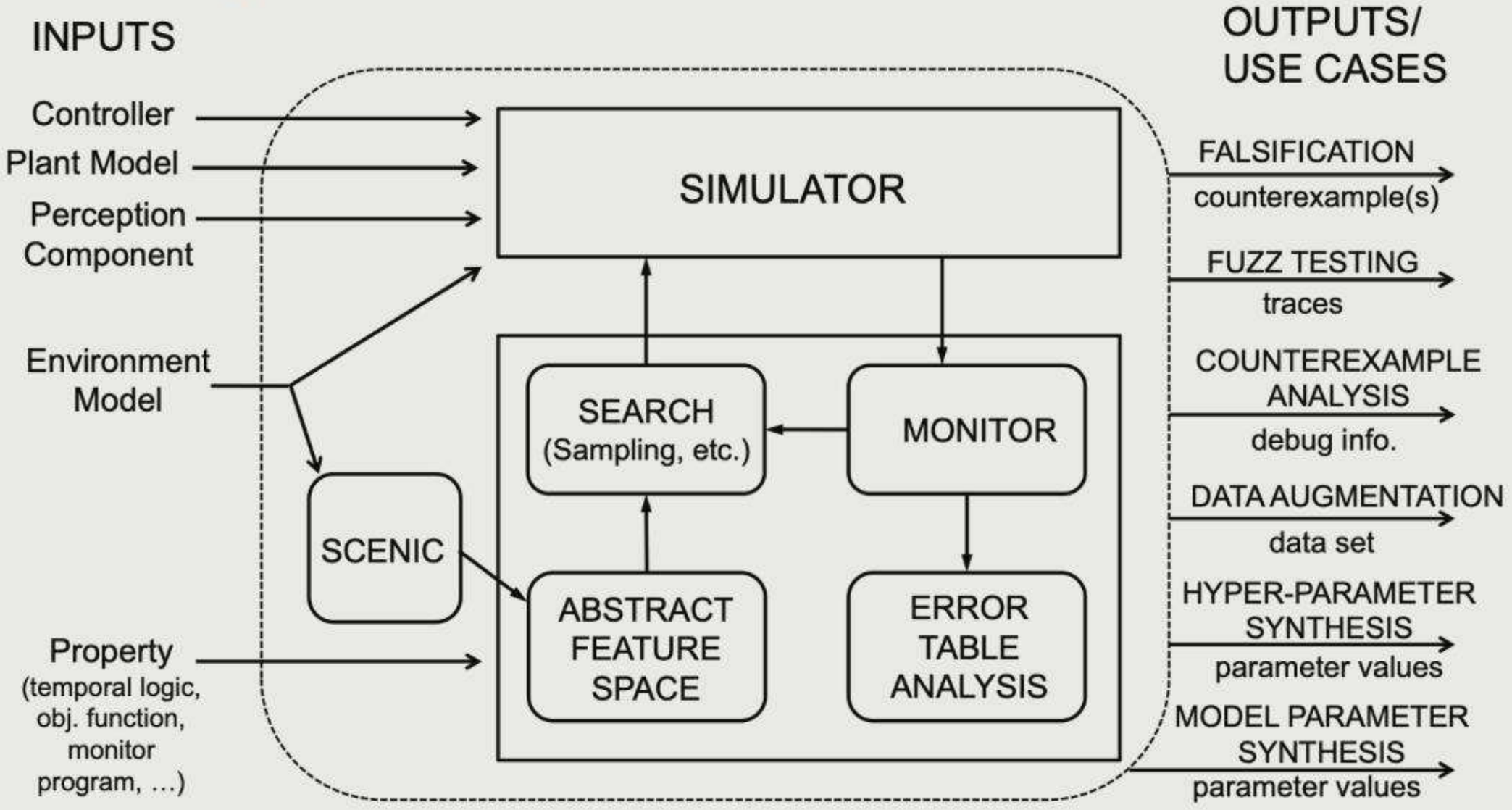
- Fundamentally new problem in CS → rich possibilities at all levels
- Intelligent data generation for a variety of data types
- Environment modeling, including modeling dynamic agents
- Generalized theory of CI enabling new applications

- Verifying ML/AI-based systems: the VerifAI toolkit [CAV 2019]

VERIFAI: A Toolkit for the Design and Analysis of AI-Based Systems

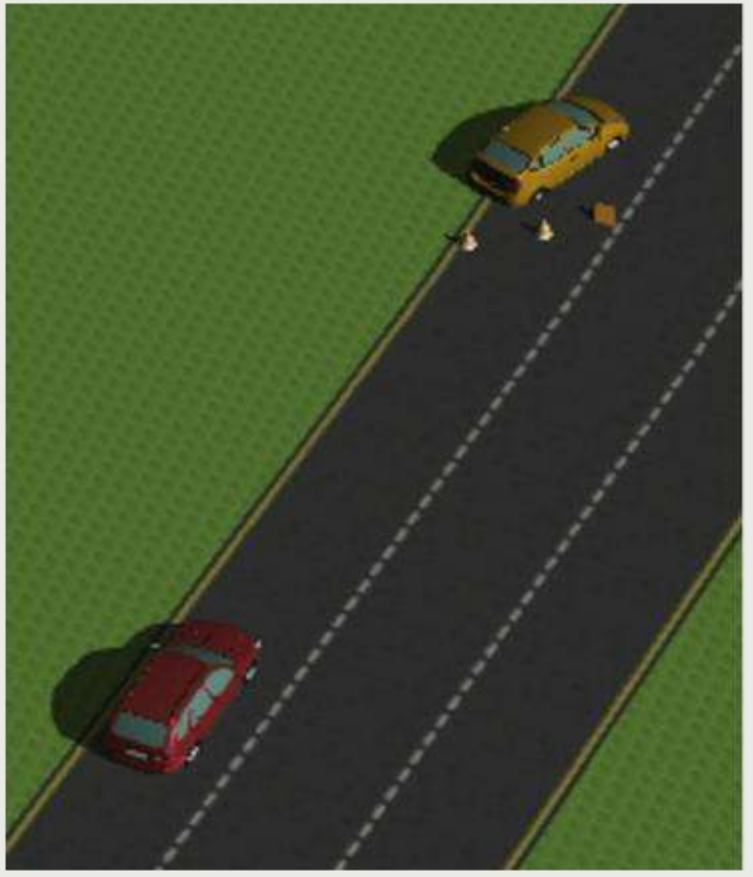
[CAV'19]

https://github.com/BerkeleyLearnVerify/VerifAI

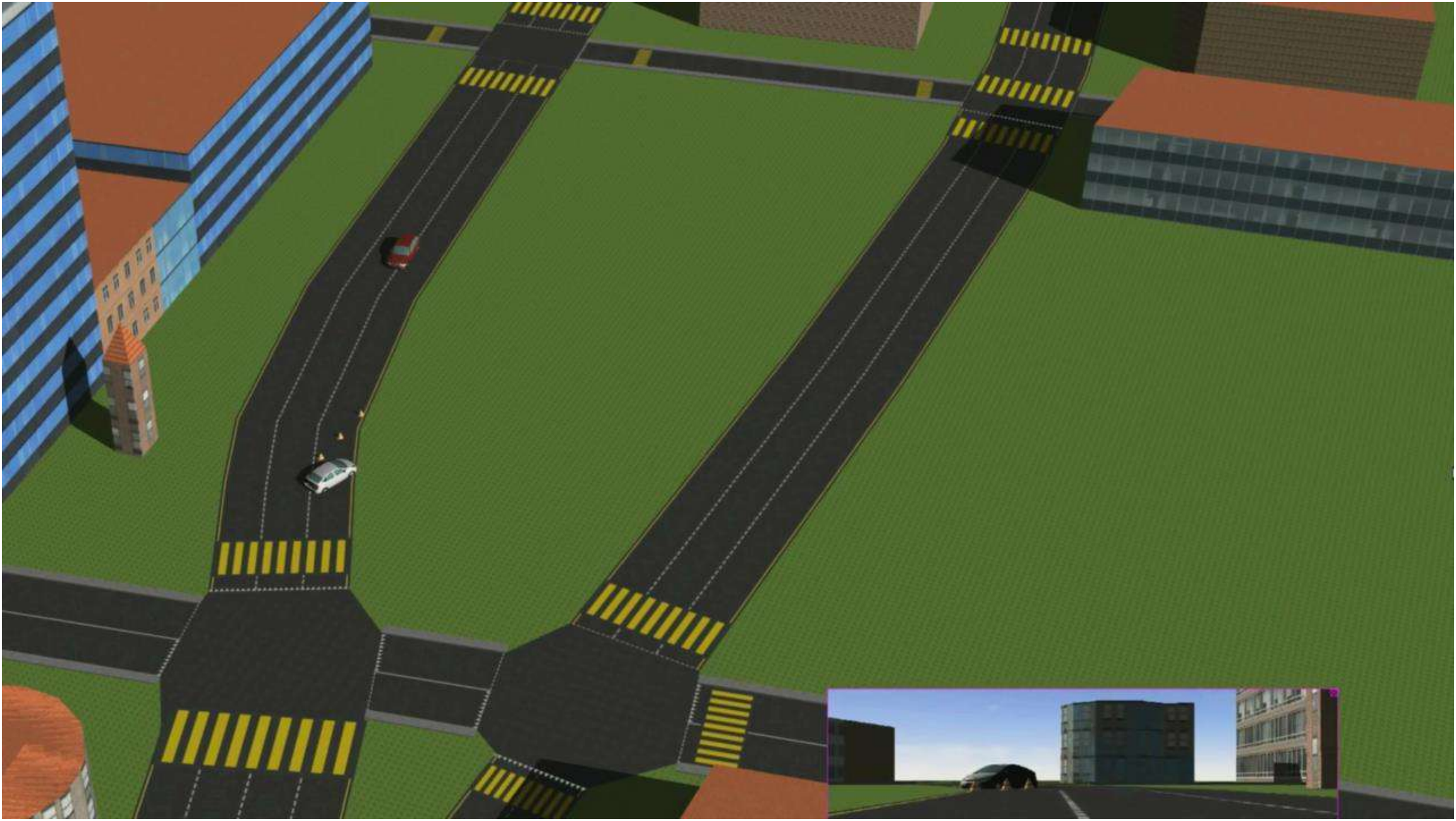


Modeling Case Study in the SCENIC Language

```
# Pick location for blockage randomly along curb
blockageSite = OrientedPoint on curb

# Place traffic cones
spot1 = OrientedPoint left of blockageSite by (0.3, 1)
cone1 = TrafficCone at spot1,
                     facing (0, 360) deg

...

# Place disabled car ahead of cones
SmallCar ahead of spot2 by (-1, 0.5) @ (4, 10),
         facing (0, 360) deg
```



Fremont et al., *Scenic: A Language for Scenario Specification and Scene Generation*, PLDI 2019 (to appear).



The Future of Algorithmic Improvisation

- Fundamentally new problem in CS → rich possibilities at all levels
- Intelligent data generation for a variety of data types
- Environment modeling, including modeling dynamic agents
- Generalized theory of CI enabling new applications
- Verifying ML/AI-based systems: the VerifAI toolkit

https://github.com/BerkeleyLearnVerify/VerifAI

Thank you!