Formal Synthesis: Oracle-guided Learning of Compositional Concepts

Susmit Jha
Principal Scientist
Computer Science Laboratory
SRI International

Acknowledgement: Sanjit A. Seshia, other fellow co-travelers over the last decade, my employers Intel, Raytheon Technologies (UTRC) and SRI, and US funding agencies DARPA, IARPA, NSA, NSF and ARL.

Formal Synthesis

- Formal synthesis is the process of learning a compositional concept satisfying a high-level formal specification.

- Programs, controllers, and explanations/intent of machine learning models/agents are examples of such compositional concepts.

- Compositionality enables automated deduction – making it no longer a pure learning problem.

- Over the last decade, we have developed several techniques for formal synthesis that include:
    - Programs: ICSE’10 (MIP Award at ICSE’20), PLDI’11, DTTC’13, NSV’14, Acta Informatica’17
    - Controllers: ICCPS’10, EMSOFT’11, IJBRA’12, FORMATS’16, FORMATS’18, Allerton’18, ACC’19
    - Explanations and intent: NFM’17, RV’17, NFM’18, JAR’18, NeurIPS’18, FMSD’19
    - Fun: A Case Study on Automated Synthesis of Magic Card Tricks. Jha et al. IEEE Formal Methods in Computer-Aided Design (FMCAD), 2016

Formal Synthesis: Talk Outline

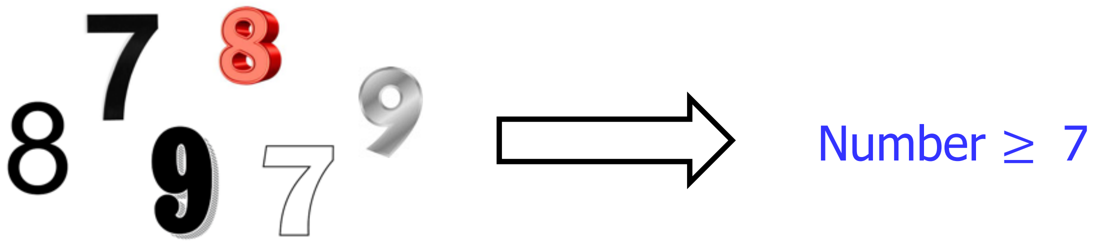
- A unifying view that describes formal synthesis techniques as an interaction between an oracle and a learner – both of which are co-designed for a target concept.

- A theoretical characterization of different formal synthesis techniques by considering oracles and learners with different properties.
  - Acta Informatica'17

- Example practical applications of this framework
  - Jha et al. Oracle-guided component-based program synthesis. ACM/IEEE International Conference on Software Engineering (ICSE), 2010 (10 year MIP award at ICSE'2020)
  - Jha et al. Explaining AI Decisions Using Efficient Methods for Learning Sparse Boolean Formulae. Journal of Automated Reasoning, 2018
  - Jha et al. Data-efficient Learning of Robust Control Policies. Allerton Control, 2018

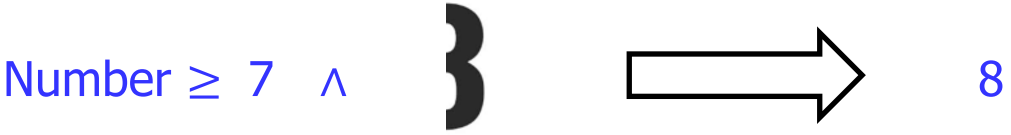
Why Formal Synthesis? Induction + Deduction

- Induction: Inferring general rules (functions) from specific examples (observations)
  - Generalization



Number ≥ 7

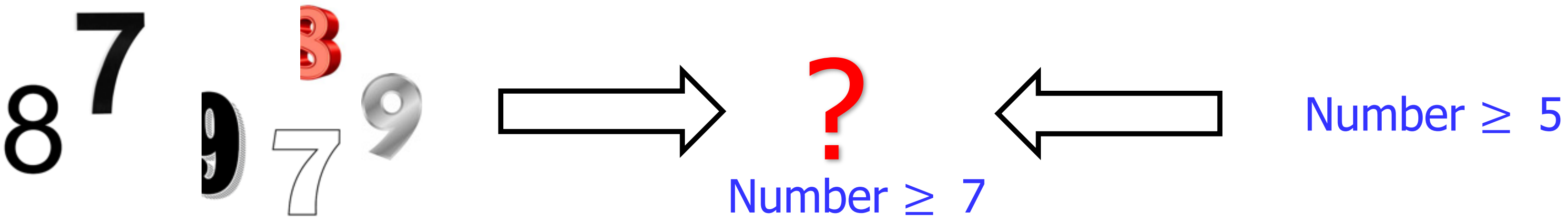
- Deduction: Applying general rules to derive conclusions about specific instances
  - Specialization



Number ≥ 7 ∧

8

- Formal Synthesis: Simultaneously learn from data and deduce from rules.



?

Number ≥ 5

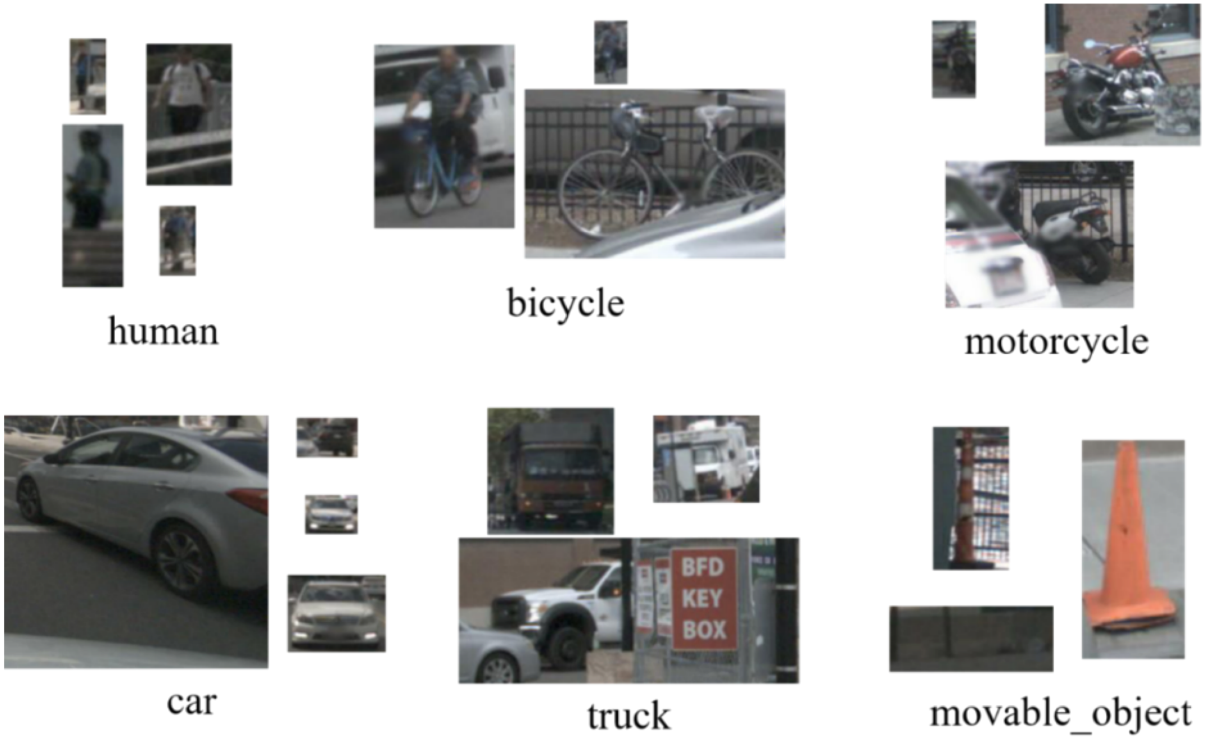
Number ≥ 7

"Towards Automatic System Synthesis Using Sciduction – Structurally Constrained Induction and Deduction." Susmit Jha, Ph.D. Thesis (UC Berkeley, 2011).

E.g. Perception as Synthesis vs Inductive Learning

NuScenes dataset

Object classes and frequency of samples: human (19.46%), **bicycle (1.04%), motorcycle (1.11%)**, car (43.62%), truck (12.70%), movable_object (22.05%)



Occlusion at different levels

E.g. Perception as Synthesis vs Inductive Learning

NuScenes dataset

Object classes and frequency of samples: human (19.46%), **bicycle (1.04%), motorcycle (1.11%)**, car (43.62%), truck (12.70%), movable_object (22.05%)

+ Spatio-temporal rules of co-occurrence in addition to data.

Model	Occlusion (%)	Overall accuracy	Class-wise accuracy					
			human	bicycle	motor-cycle	car	truck	movable object
CNN - ResNet (Baseline)	No occlusion	88.65	92.44	57.24	61.31	92.59	69.74	90.69
CNN - ResNet (Baseline)	30%	83.24	90.99	12.52	20.90	92.48	71.15	71.36
CNN - ResNet (Baseline)	50%	79.17	90.93	2.36	12.48	87.33	58.94	67.95
Formal Synthesis	No occlusion	95.51	98.38	66.25	73.37	97.13	82.17	98.62
Formal Synthesis	30%	94.70	97.72	65.66	65.40	96.62	81.31	96.73
Formal Synthesis	50%	93.13	97.53	31.36	64.88	94.17	81.10	96.34

Less frequent (~1%) classes

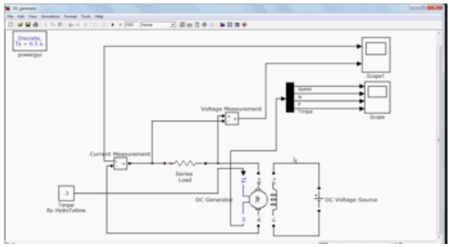
Formal Synthesis: Why Oracle-guided?

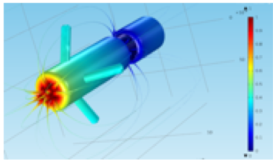
(inp_1, op_1)
(inp_2, op_2)
… … …
(inp_N, op_N)

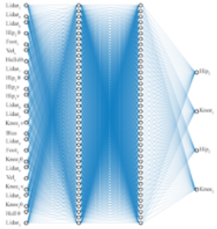
```
int deobfuscated (int y)
{   z = y << 2;
    y = z + y;
    z = y << 3;
    y = z + y;
    return y;   }
```

```
int obfuscated (int y)
{ a=1; b=0; z=1; c=0;
  while(1) {
    if (a == 0) {
    if (b == 0) { y=z+y; a =~a;
    b=~b;  c=~c; if (~c) break; }
   else {
     z=z+y; a=~a; b=~b; c=~c;
     if (~c) break; } }
    else if(b == 0) {z=y << 2;
a=~a;}
     else { z=y << 3; a=~a; b=~b;}
}}
```

Even when symbolic specification is available,
it may not be amenable to deduction.







Specification is a black-box oracle and
not available in symbolic form.

E.g. static program analysis is challenging and obfuscation can target these limitations, interaction with humans, deep learning models too large to be considered as white-box, physics models too complex to be analytically represented for deduction.

Oracle Interfaces beyond positive and negative examples

Oracle Interfaces

LEARNER

Positive Witness

$x \in \phi$, if one exists, else $\perp$

Negative Witness

$x \notin \phi$, if one exists, else $\perp$

Membership: Is $x \in \phi$?

Yes / No

Equivalence: Is $f = \phi$?

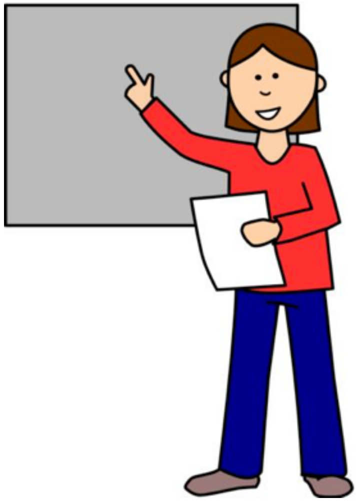
Yes / No + $x \in \phi \oplus f$

Subsumption/Subset: Is $f \subseteq \phi$?

Yes / No + $x \in f \setminus \phi$

Distinguishing Input: f, $X \subseteq f$

f' s.t. $f' \neq f \wedge X \subseteq f'$, if it exists; o.w. $\perp$



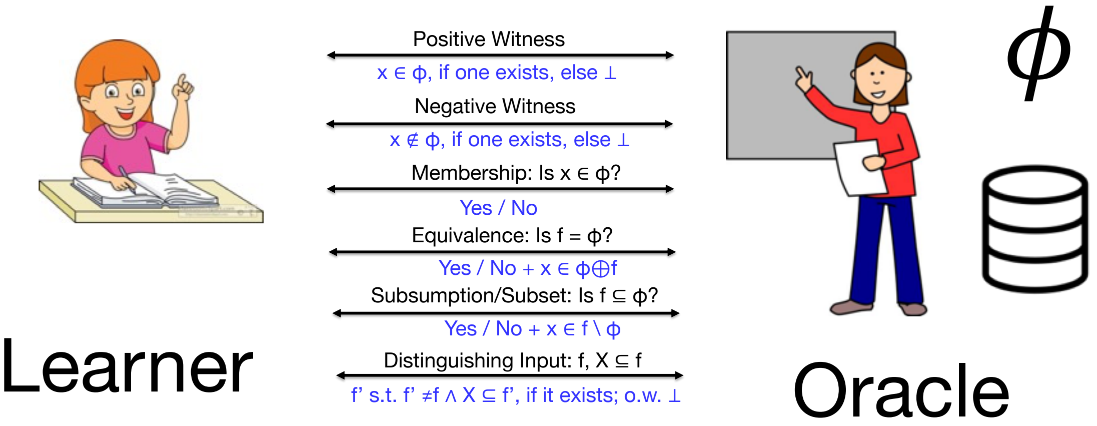
ORACLE

Oracle Guided Formal Synthesis

Oracle Interface

$\mathcal{C}$

Concept Class



Positive Witness

$x \in \phi$, if one exists, else $\perp$

Negative Witness

$x \notin \phi$, if one exists, else $\perp$

Membership: Is $x \in \phi$?

Yes / No

Equivalence: Is $f = \phi$?

Yes / No + $x \in \phi \oplus f$

Subsumption/Subset: Is $f \subseteq \phi$?

Yes / No + $x \in f \setminus \phi$

Distinguishing Input: $f, X \subseteq f$

f' s.t. $f' \neq f \wedge X \subseteq f'$, if it exists; o.w. $\perp$

Learner

Oracle

ϕ

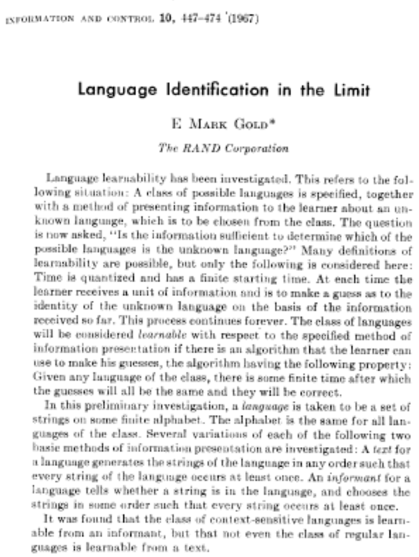
A dialogue is a sequence of (query, response) confirming to an oracle interface O

An Oracle-guided formal synthesis algorithm is a pair <L, T> where

- L is a learner, a non-deterministic algorithm mapping a dialogue to a concept c and query q
- T is an oracle/teacher, a non-deterministic algorithm mapping a dialogue and query to a response r

An Oracle-guided formal synthesis algorithm <L,T> solves a synthesis problem if there exists a dialogue between L and T that converges in the target concept f ∊ C

Oracle guided formal synthesis as a generalization of query based learning



INFORMATION AND CONTROL **10**, 447–474 (1967)

Language Identification in the Limit

E Mark Gold*

The RAND Corporation

Language learnability has been investigated. This refers to the following situation: A class of possible languages is specified, together with a method of presenting information to the learner about an unknown language, which is to be chosen from the class. The question is now asked, "Is the information sufficient to determine which of the possible languages is the unknown language?" Many definitions of learnability are possible, but only the following is considered here: Time is quantized and has a finite starting time. At each time the learner receives a unit of information and is to make a guess as to the identity of the unknown language on the basis of the information received so far. This process continues forever. The class of languages will be considered *learnable* with respect to the specified method of information presentation if there is an algorithm that the learner can use to make his guesses, the algorithm having the following property: Given any language of the class, there is some finite time after which the guesses will all be the same and they will be correct.

In this preliminary investigation, a *language* is taken to be a set of strings on some finite alphabet. The alphabet is the same for all languages of the class. Several variations of each of the following two basic methods of information presentation are investigated: A *text* for a language generates the strings of the language in any order such that every string of the language occurs at least once. An *informant* for a language tells whether a string is in the language, and chooses the strings in some order such that every string occurs at least once.

It was found that the class of context-sensitive languages is learnable from an informant, but that not even the class of regular languages is learnable from a text.

- Formal Language Learning, 1967
- Class of languages identifiable in the limit if there is a learning procedure that, for each language in that class, given an infinite stream of strings, will eventually generate a representation of the language.
- Sample results:
  - Cannot learn regular languages, CFLs, CSLs using just positive witness queries
  - Can learn using both positive & negative witness queries

Dana Angluin

- Queries and Concept Learning, 1988
- Supports witness, equivalence, membership queries
- Sample results:
  - Can learn DFAs in poly time from membership and equivalence queries
  - Cannot learn DFAs or DNF formulas in poly time with just equivalence queries

Use advances in automated deduction techniques such as program analysis, control theory, etc. to support richer oracle interfaces.

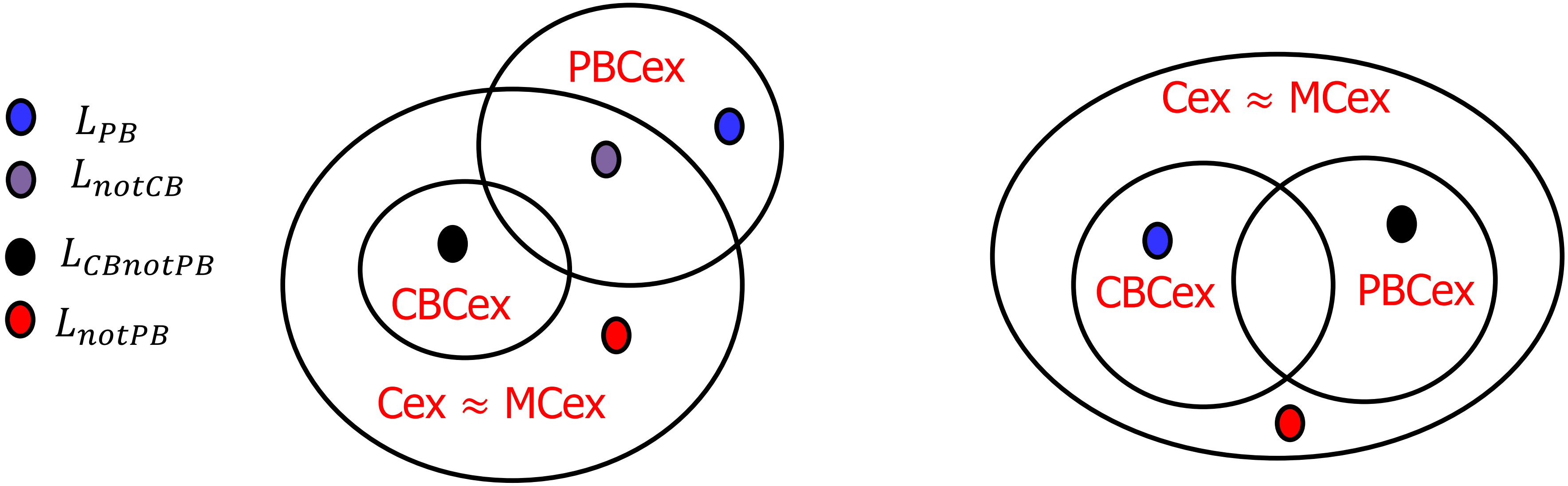
Theoretical Analysis of Oracle Guided Formal Synthesis: Jha & Seshia, Acta Informatica'17

For concept class of programs recognizing indexed family of regular languages, we studied Oracle-guided formal synthesis algorithm <L, T> for synthesis feasibility where

- Learner L has different memory:
  - Finite memory
  - Infinite memory
- Oracle O has different interfaces:
  - Cex: Arbitrary counterexamples
    - Any element of $f \oplus \varphi$
  - MCex: Minimal counterexamples
    - A least element of $f \oplus \varphi$ according to size
    - Motivated by debugging methods that seek to find small counterexamples to explain errors & repair
  - CBCex: Constant-bounded counterexamples (bound B)
    - An element x of $f \oplus \varphi$ s.t. $\text{size}(x) < B$
    - Motivation: Bounded Model Checking, Context bounded testing, etc.
  - PBCex: Positive-bounded counterexamples
    - An element x of $f \oplus \varphi$ s.t. size(x) is no larger than that of any positive example seen so far
    - Motivation: bug-finding methods that mutate a correct execution in order to find buggy behaviors

Theoretical Analysis of Oracle Guided Formal Synthesis: Jha & Seshia, Acta Informatica'17

- Cex: Arbitrary counterexamples
  - Any element of f ⊕ φ
- MCex: Minimal counterexamples
  - A least element of f ⊕ φ according to size
- CBCex: Constant-bounded counterexamples (bound B)
  - An element x of f ⊕ φ s.t. size(x) < B
- PBCex: Positive-bounded counterexamples
  - An element x of f ⊕ φ s.t. size(x) is no larger than that of any positive example seen so far

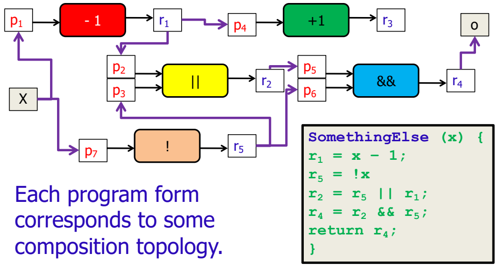


Finite memory learner

Infinite memory learner

Example 1: Program Synthesis (ICSE' 2010)

Concept Class: Programs as composition of functions.

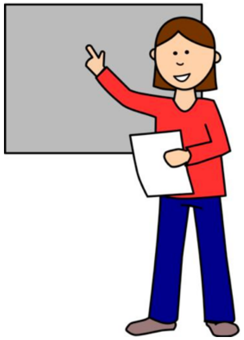


```
SomethingElse (x) {
r1 = x - 1;
r5 = !x
r2 = r5 || r1;
r4 = r2 && r5;
return r4;
}
```

Each program form corresponds to some composition topology.

Distinguishing Input: f, X ⊆ f

f' s.t. f' ≠f ∧ X ⊆ f', if it exists; o.w. ⊥



Learner

Oracle Interface

Oracle

Implemented using Satisfiability Solving

Implemented using Satisfiability Solving

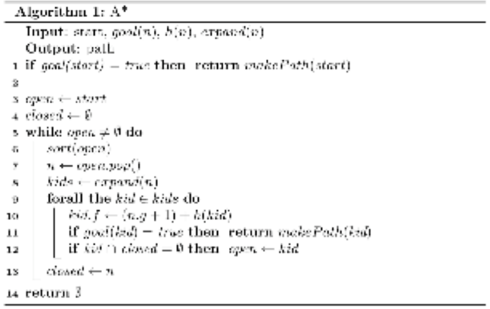
```
P24: Round up to next
    highest power of 2
o1 :=  x - 1;
o2 := o1 >> 1;
o3 := o1 || o2;
o4 :=  o3 >> 2;
o5 :=  o3 || o4;
o6 :=  o5 >> 4;
o7 :=  o5 || o6;
o8 :=  o7 >> 8;
o9 :=  o7 || o8;
o10 := o9>>16;
o11 := o9 || o10;
res := o10 && 1;
```

```
P25: Higher order half of
    product of x and y
o1 := x && 0xFFFF;
o2 := x >> 16;
o3 :=  y && 0xFFFF;
o4 := y >> 16;
o5 :=  o1 X o3;
o6 :=  o2 X o3;
o7 :=  o1 X o4;
o8 :=  o2 X o4);
o9 := o5 >> 16;
o10 :=  o6 + o9;
o11 :=  o10 && 0xFFFF;
o12 := o10 >> 16;
o13 :=  o7 + o11;
o14 :=  o13 >> 16);
o15 :=  o14 + o12;
res :=  o15 + o8;
```

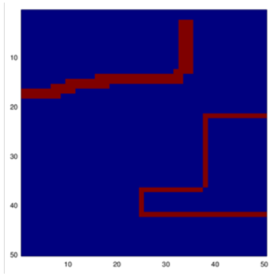
Example 2 – Explaining AI Models (NFM’17, JAR’18)

What parts of the input are relevant to property ϕ being satisfied or not?

E.g. if Bridge A is blocked, optimal path goes via Bridge B



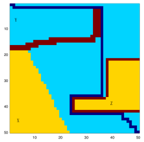
```
Algorithm 1: A*
  Input: start, goal(n), h(n), expand(n)
  Output: path
if goal(start) = true then return makePath(start)

open ← start
closed ← ∅
while open ≠ ∅ do
    sort(open)
    n ← open.pop()
    kids ← expand(n)
    forall the kid ∈ kids do
        kid.f ← (n.g + 1) + h(kid)
        if goal(kid) = true then return makePath(kid)
        if kid ∩ closed = ∅ then open ← kid
    closed ← n
return ∅
```

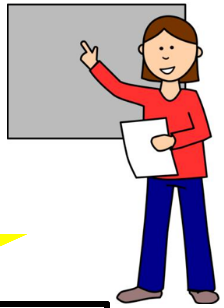


Membership: Is x ∈ ϕ?

Yes / No

Oracle Interface





Does plan satisfy ϕ
E.g. Plan goes via some segment

Learner

Oracle

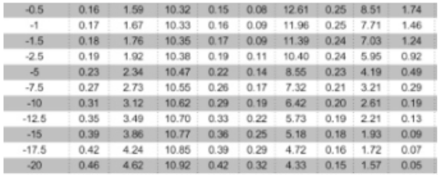
Design of experiments (input assignments)
m1 = (0,0,0,1,1,0,1)
m2 = (0,0,1,1,0,1,0)

O1 = (0)
O2 = (1)

Example 2 – Explaining AI Models (NFM’17, JAR’18)

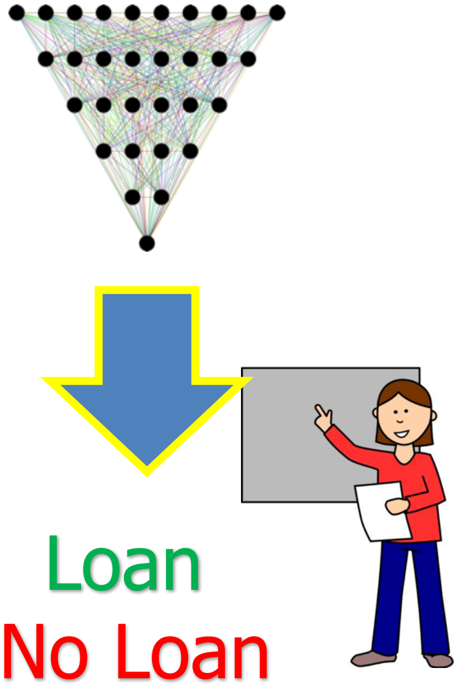
What parts of the input are relevant to property ϕ being satisfied or not?

E.g. does race play a role in loan eligibility decision?



-0.5	0.16	1.59	10.32	0.15	0.08	12.61	0.25	8.51	1.74
-1	0.17	1.67	10.33	0.16	0.09	11.96	0.25	7.71	1.46
-1.5	0.18	1.76	10.35	0.17	0.09	11.39	0.24	7.03	1.24
-2.5	0.19	1.92	10.38	0.19	0.11	10.40	0.24	5.95	0.92
-5	0.23	2.34	10.47	0.22	0.14	8.55	0.23	4.19	0.49
-7.5	0.27	2.73	10.55	0.26	0.17	7.32	0.21	3.21	0.29
-10	0.31	3.12	10.62	0.29	0.19	6.42	0.20	2.61	0.19
-12.5	0.35	3.49	10.70	0.33	0.22	5.73	0.19	2.21	0.13
-15	0.39	3.86	10.77	0.36	0.25	5.18	0.18	1.93	0.09
-17.5	0.42	4.24	10.85	0.39	0.29	4.72	0.16	1.72	0.07
-20	0.46	4.62	10.92	0.42	0.32	4.33	0.15	1.57	0.05

Individual Data



Membership: Is x ∈ ϕ?

Yes / No

Oracle Interface

Loan

No Loan

Learner

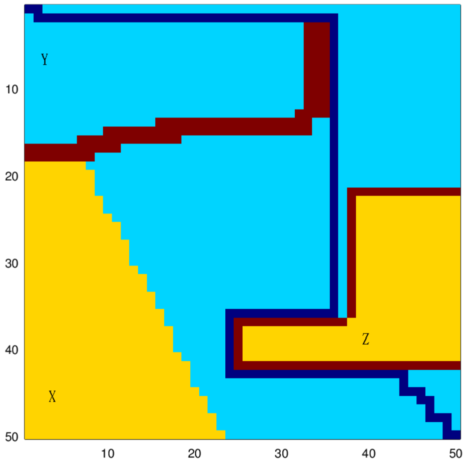
Oracle

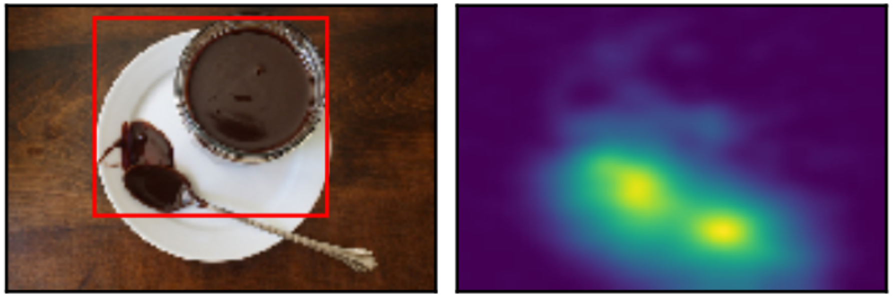
Design of experiments (input assignments)
m1 = (0,0,0,1,1,0,1)
m2 = (0,0,1,1,0,1,0)

O1 = (0)
O2 = (1)

Example 2 – Explaining AI Models (NFM’17, JAR’18)

Explanations of decisions are often short and involve only few variables !

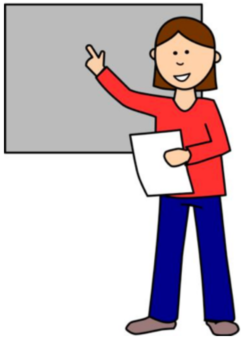




Example 2 – Explaining AI Models (NFM’17, JAR’18)

Membership: Is x ∈ ϕ?

Yes / No



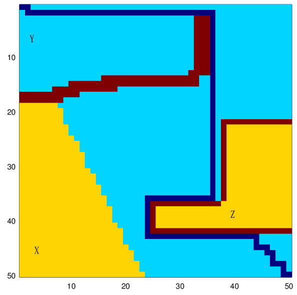
Learner

Implemented using Hamming Distance Search over inputs/instances

Oracle Interface

Oracle

Implemented using Local Verifier/Evaluator

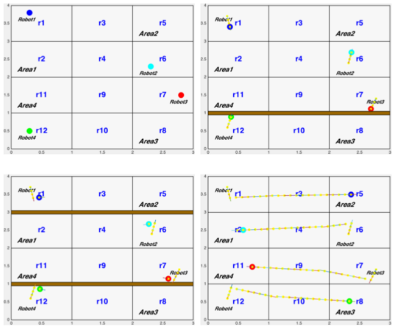


Explaining A* Planning

|V| = 2500

|U| <= 4

Runtime < 3 minutes

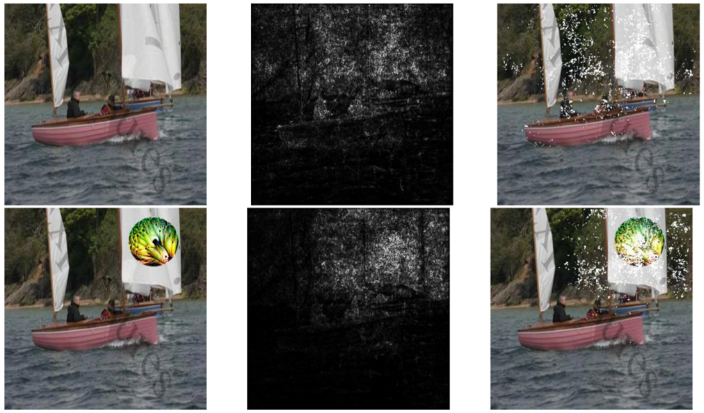


Reactive Exploration Strategy

|V| = 96

|U| <= 2

Runtime < 5 seconds



(Jha et al. NeurIPS’19: Confidence, Detecting adv attacks)

Example 2 – Explaining AI Models (NFM'17, JAR'18)

V : set of all variables U : set of relevant variables

- Randomly sample till black box differs on two assignments or a bound M is reached.

$M = 2^{|U|}\, ln\left(\frac{1}{1-\kappa}\right)$

for κ PAC guarantee

- Binary Search over Hamming Distance between two assignments to find a relevant variable.

$ln(|V|)$

- Add this relevant variable to the relevant set. Recursively find relevant variables from the remaining input set for each possible assignment to this relevant variable.

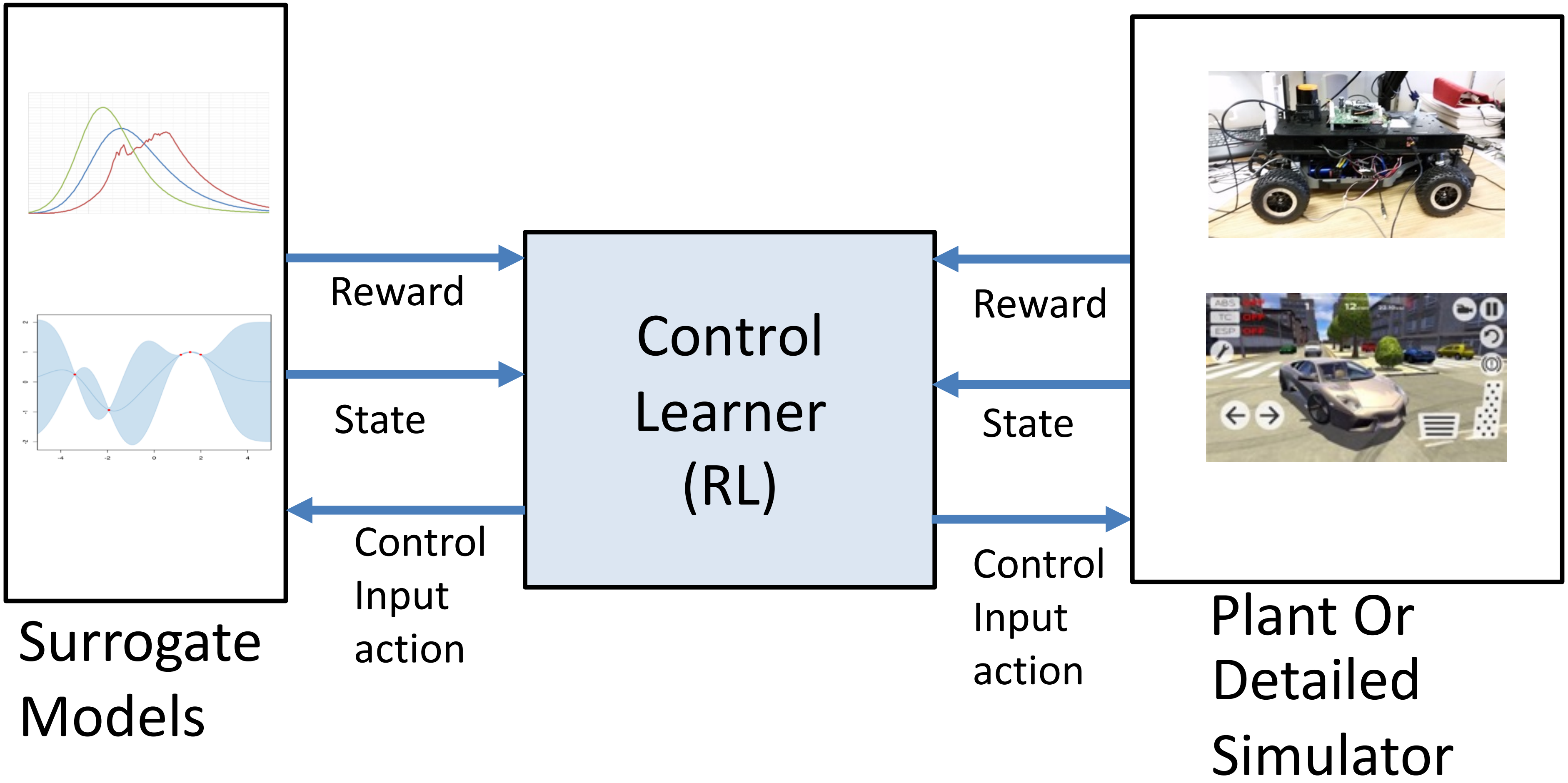
$2^{|U|}$

Relevant variables can be found with confidence $\boldsymbol{\kappa}$ in

$$\mathbf{2^{2|U|}}\boldsymbol{ln(|V|/(1-\kappa))}$$

queries to the oracle.

Example 3: Hierarchical Oracles - Controller Synthesis (Jha and Lincoln, Allerton Control’18)



Reward

State

Control Input action

Control Learner (RL)

Reward

State

Control Input action

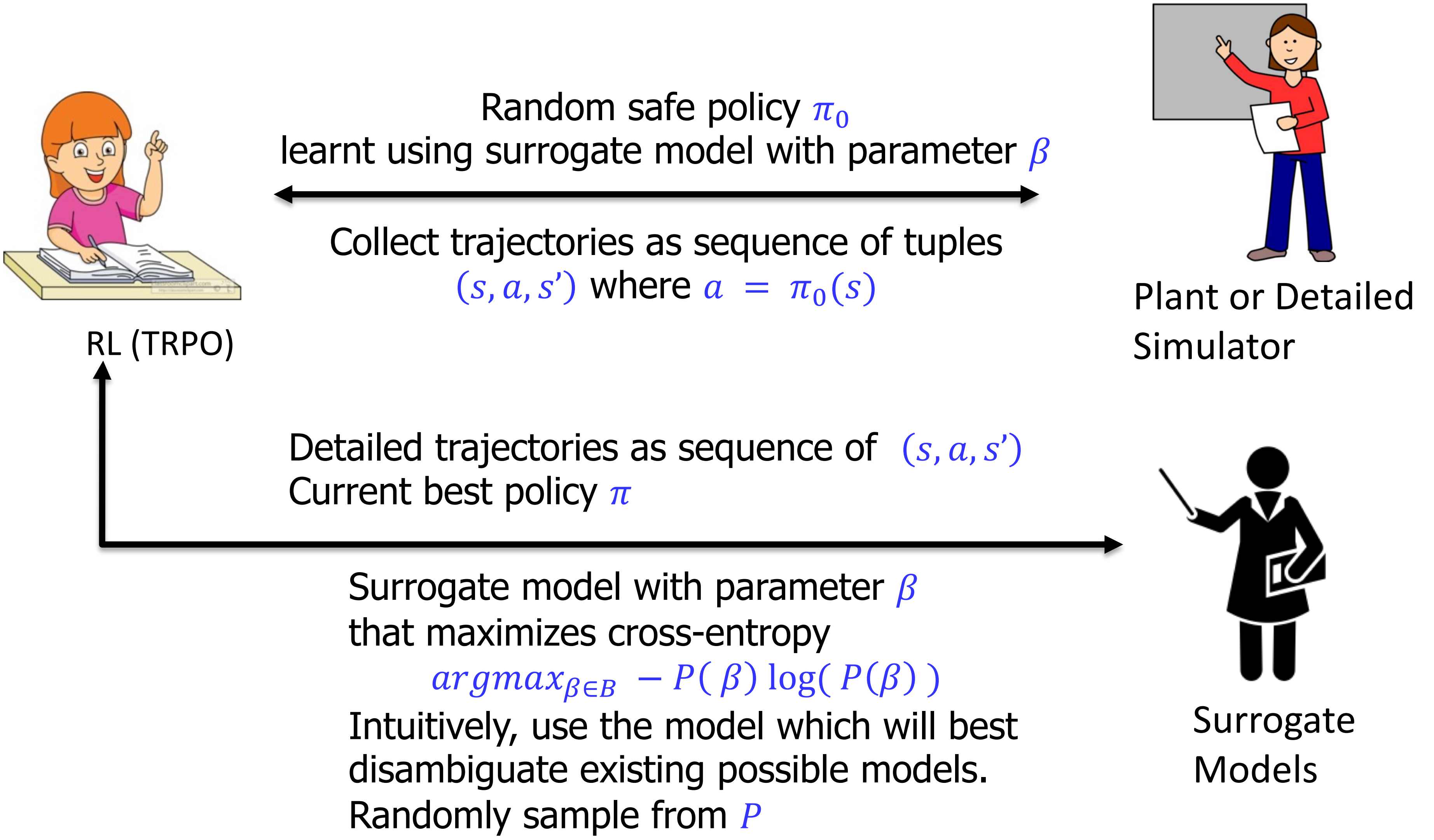
Surrogate Models

Plant Or Detailed Simulator

Direct experiments on real world are costly, slow, and risk-prone.

Minimize the number of real world trajectories needed to learn the optimal policy and simultaneously tune the surrogate model being used by RL

Example 3: Hierarchical Oracles - Controller Synthesis (Jha and Lincoln, Allerton Control'18)



Random safe policy π_0 learnt using surrogate model with parameter β

Collect trajectories as sequence of tuples (s, a, s') where $a = \pi_0(s)$

Plant or Detailed Simulator

RL (TRPO)

Detailed trajectories as sequence of (s, a, s')
Current best policy π

Surrogate model with parameter β that maximizes cross-entropy

$$argmax_{\beta \in B} \; - P(\beta) \log(P(\beta))$$

Intuitively, use the model which will best disambiguate existing possible models.
Randomly sample from P

Surrogate Models

Guarantee of convergence for TRPO or any RL method that guarantees the KL divergence between consecutive policies is bounded.

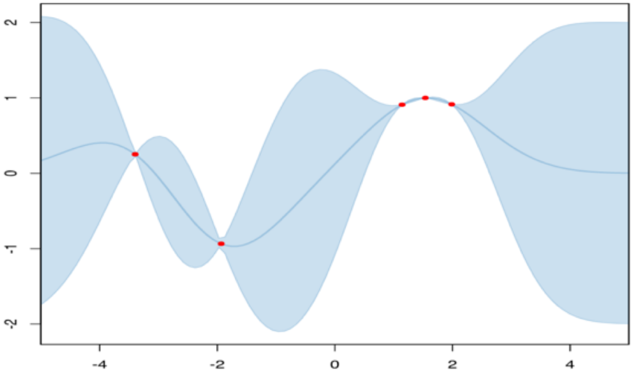
Example 3: Hierarchical Oracles - Controller Synthesis (Jha and Lincoln, Allerton Control'18)

Random pick parameter β from parameter space B
Simulate physics engine model for state-action pair (s, a) to get $(s, a, F(s, a, \beta))$

For each sampled β, we can compute average model-deviation as:

$$\Delta(\beta) = \frac{1}{|\mathcal{T}|} \sum_{(s,a,s')\in\mathcal{T}} ||s' - F(s,a,\beta)||_2$$

Surrogate Models



Training data: $(\beta_0, \Delta(\beta_0)), (\beta_1, \Delta(\beta_1)), (\beta_2, \Delta(\beta_2)), \ldots$

Learn a Gaussian Process approximation of the function Δ [Squared Exponential Kernel]

Typical use of GPs to learn functions is often to find the extremum point (maximum/minimum). But for us Δ itself is not fixed and will change as we collect more real-world trajectories. Our interest is, thus, in maintaining a probability distribution P over how likely a parameter β is to correspond to the most accurate model of the real world.

Example 3: Hierarchical Oracles - Controller Synthesis (Jha and Lincoln, Allerton Control’18)

$$\Delta(\beta) \ = \frac{1}{|\mathcal{T}|} \sum_{(s,a,s')\in\mathcal{T}} \left|\left|s' - F(s,a,\beta)\right|\right|_2$$

For each $\beta \in B$, we define an indicator whether β minimizes Δ :

$$\delta(\Delta, \beta) = 1 \ if \ \beta = argmin_b \ \Delta(b)$$

Now, the probability that β is the optimal model parameter representing the plant is given by:

$$P(\beta) = \int_\Delta p_{\mu,\sigma}(\Delta) \cdot \delta(\Delta, \beta) \, d\Delta$$

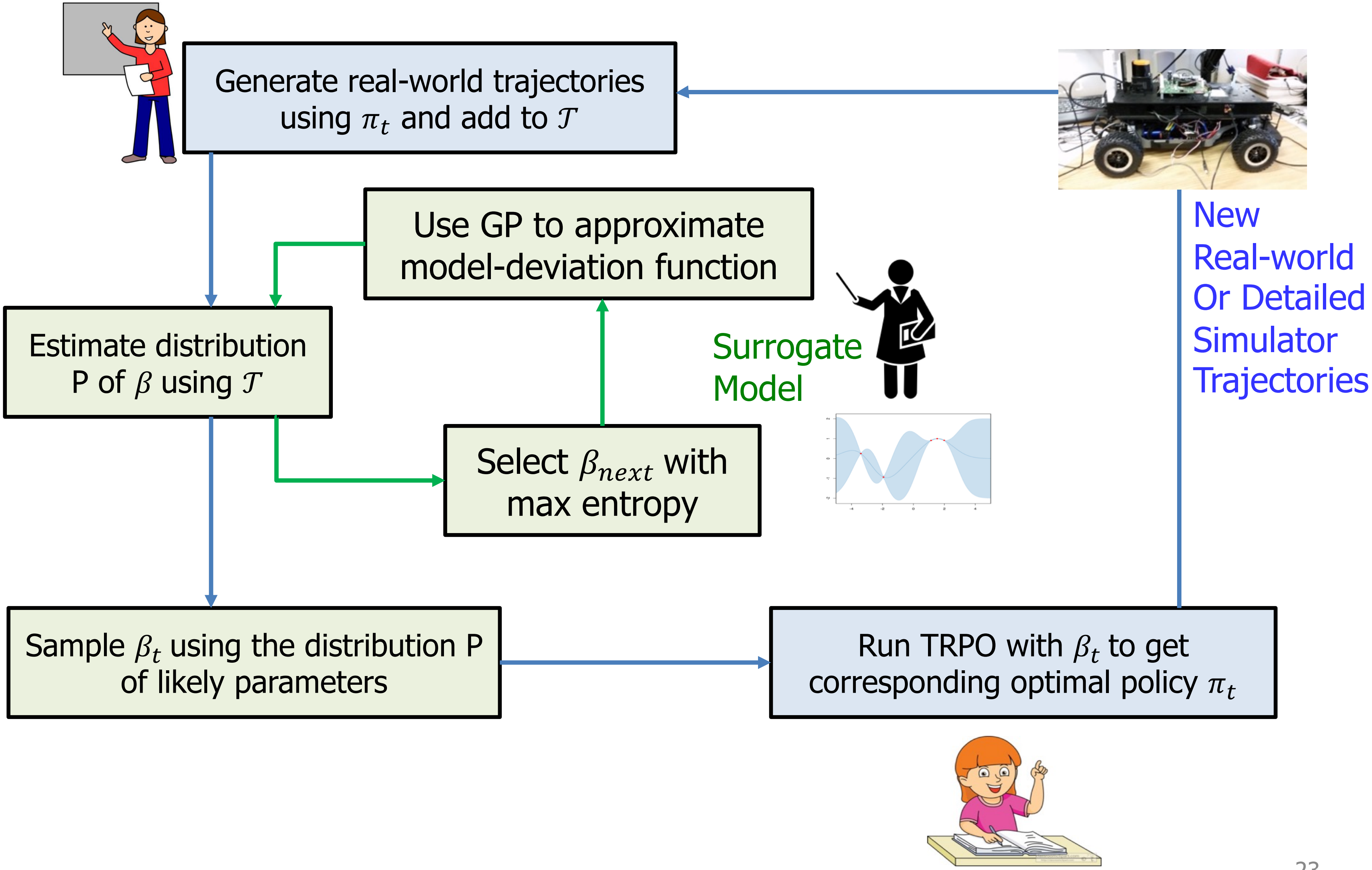
The probability of β is the sum of probability of model-deviation functions in which β minimizes these functions.

How do we compute this? Perform Monte Carlo sampling to approximate the revised distribution P. Sample $\Delta_1, \Delta_2, \Delta_3 \dots, \Delta_N$ from the GP, for each $\beta \in B$, we compute the number of times r that β minimizes Δ_i

[this is just evaluation, no simulation or real-world experiment]

$$P(\beta) = \frac{r}{N}$$

Example 3: Hierarchical Oracles - Controller Synthesis (Jha and Lincoln, Allerton Control'18)



Generate real-world trajectories using π_t and add to $\mathcal{T}$

Estimate distribution P of β using $\mathcal{T}$

Use GP to approximate model-deviation function

Select β_{next} with max entropy

Surrogate Model

Sample β_t using the distribution P of likely parameters

Run TRPO with β_t to get corresponding optimal policy π_t

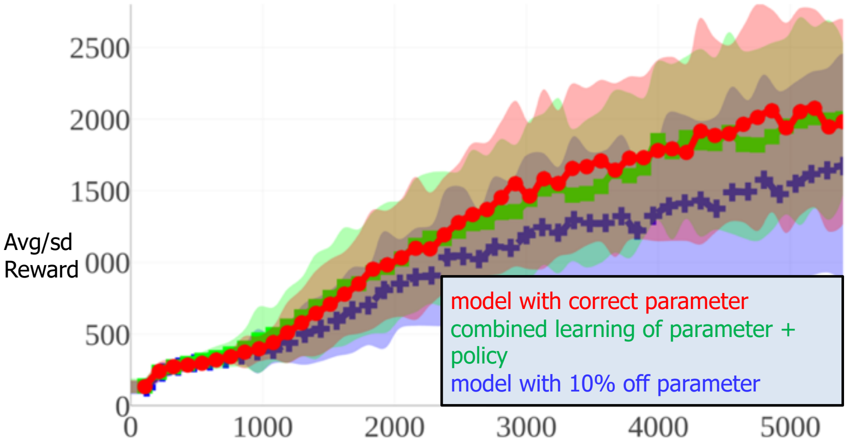
New Real-world Or Detailed Simulator Trajectories

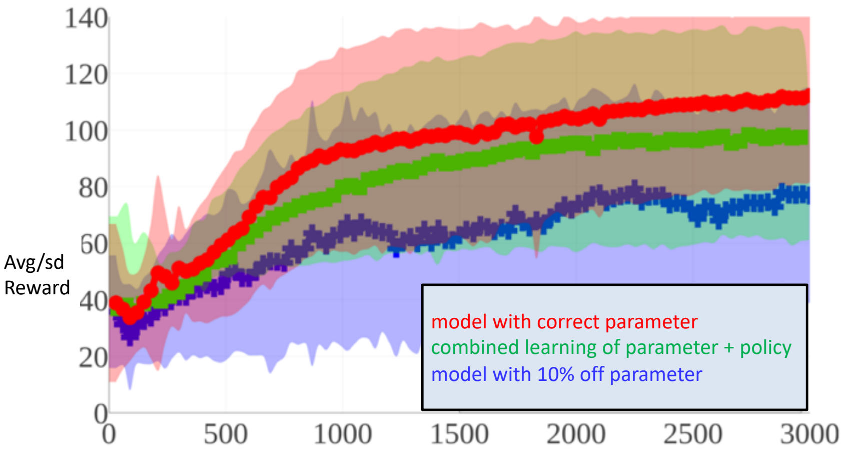
Example 3: Hierarchical Oracles - Controller Synthesis (Jha and Lincoln, Allerton Control'18)

OpenAI Gym with the MuJoCo simulator. Parameter space:

- Inverted Pendulum (IP): A pendulum is connected to a cart, which moves linearly. The dimension is 2, one for the mass of the pendulum and one for the cart.
- Swimmer: The swimmer is a 3-link planar robot and has 3 dimensions, one for the mass of each link..
- Hopper: The hopper is a 4-link planar mono-pod robot. Dimensionality: 4
- Walker2D: The walker is a 7-link planar biped robot. Dimensionality: 7
- HalfCheetah: The halfcheetah is a 7-link planar cheetah robot. Dimensionality: 7

Simulation model was constructed by decreasing the mass value by 10%.
Original OpenAI Gym model treated as real world.





Conclusion

- Formal synthesis is the process of learning a compositional concept satisfying a high-level formal specification.

- Programs, controllers, and explanations/intent of machine learning models/agents are examples of such compositional concepts, and we presented example formal synthesis approaches for these.

- A unifying view that describes formal synthesis techniques as an interaction protocol between an oracle and a learner – both of which are co-designed for a target concept.
    - A theoretical characterization of different formal synthesis techniques by considering oracles and learners with different properties.

Thanks!

References

- Jha et al. Synthesizing switching logic for safety and dwell-time requirements. ACM/IEEE International Conference on Cyber-physical Systems (ICCPS), 2010
- Jha et al. Oracle-guided component-based program synthesis. ACM/IEEE International Conference on Software Engineering (ICSE), 2010 (10 year MIP award at ICSE'2020)
- Jha et al. Synthesis of optimal switching logic for hybrid systems. ACM SIGBED International Conference on Embedded Software (EMSOFT), 2011
- Jha et al. Synthesis of insulin pump controllers from safety specifications using Bayesian model validation. Journal of Bioinformatics Research and Applications (IJBRA), 2012
- Jha et al. Synthesis of Optimal Fixed-Point Implementation of Numerical Software Routines. Numerical Software Verification (NSV), 2014
- Jha et al. On exists-forall-exists Solving: A Case Study on Automated Synthesis of Magic Card Tricks. IEEE Formal Methods in Computer-Aided Design (FMCAD), 2016
- Jha et al. TeLEx: Passive STL Learning Using Only Positive Examples. International Conference on Runtime Verification (RV), 2017
- Jha et al. On Learning Sparse Boolean Formulae For Explaining AI Decisions. NASA Formal Methods (NFM), 2017
- Jha et al. Explaining AI Decisions Using Efficient Methods for Learning Sparse Boolean Formulae. Journal of Automated Reasoning, 2018
- Jha et al. Data-efficient Learning of Robust Control Policies. Allerton Control, 2018
- Vazquez-Chanlatte et al. Learning Task Specifications from Demonstrations. Neural Information Processing Systems (NeurIPS), 2018
- Jha et al. Inferring and Conveying Intentionality: Beyond Numerical Rewards to Logical Intentions. AAAI Spring Symposium, Towards Conscious AI Systems, 2019
- Jha et al. TeLEx: Learning signal temporal logic from positive examples using tightness metric. Formal Methods in System Design (FMSD), 2019