Qualifying Exam Defense: Implicit Invariants for Relational Data Structures

Meriel Stein

May 18, 2020

Invariants

- Truisms that hold over the lifetime of a program
- Help to characterize system implementation at a higher-level
- Applications: check for correctness, find opportunities for optimization, can be monitored at runtime to check for violations and/or enforce system properties...

```
def someFunction(x):
    y = 2
    array1 = [[2 0], [0 0]]
    array2 = [x y]
    while x < 100:
        if(x<=10):
            y++
            array1[0][0] ++
        x++
    return y
```

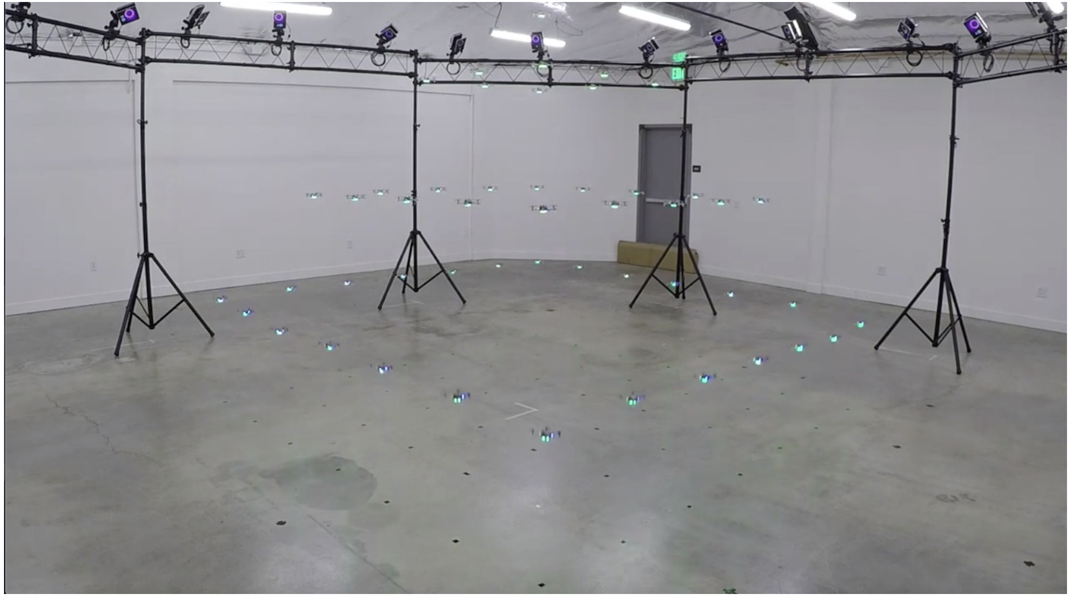
WHILE LOOP INVARIANTS:
x < ◯x
x <=10.0 → y > 2
array1[0][0] == array2[1]
array2[1] == y

POSTCONDITION INVARIANTS:
y <= 12

Relational Data Structures

- House values that are relational in placement w.r.t. other adjacent values or the indices in which they are placed
  - E.g. Tensors, 1D arrays, 2D arrays, point clouds, sets, lists...
- More interesting if they are mutable & numerically typed
  - More likely to exhibit complex behavior
  - More likely to introduce bugs?

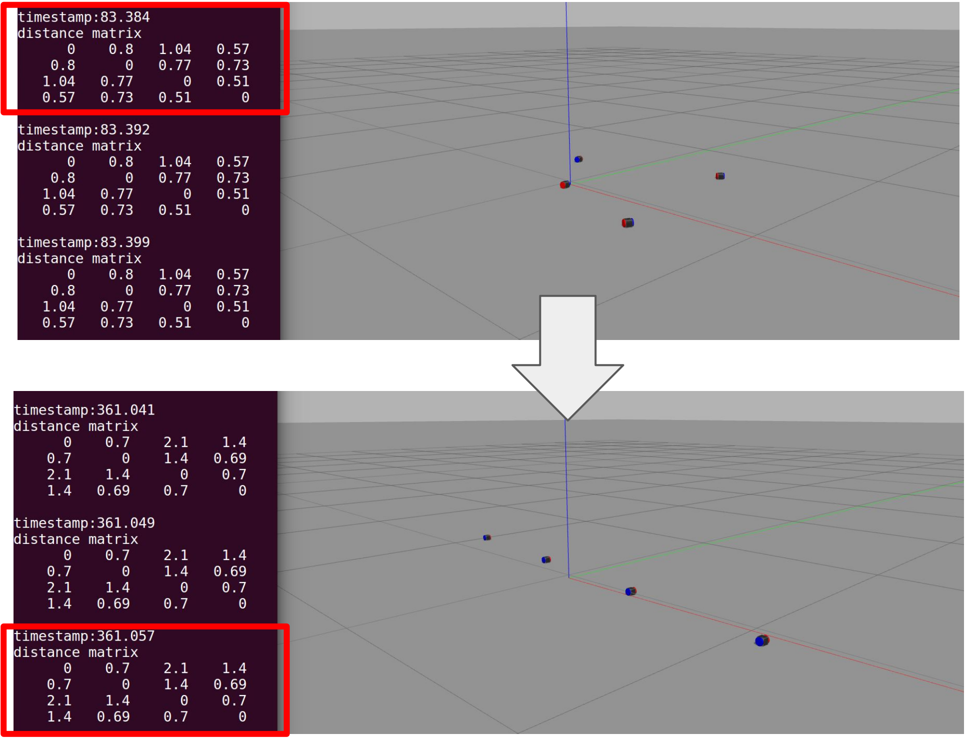
Motivation



https://www.youtube.com/watch?v=ezTayb76x9U

- Invariants for relational data structures have stronger guarantee of appearing in swarms
- Actions/states of individual members are often defined in relation to rest of swarm
- ROS messages have velocity vectors, point clouds and arrays from laser scans and other sensors, matrices representing occupancy grid maps...

Motivating Example — Ground Swarm



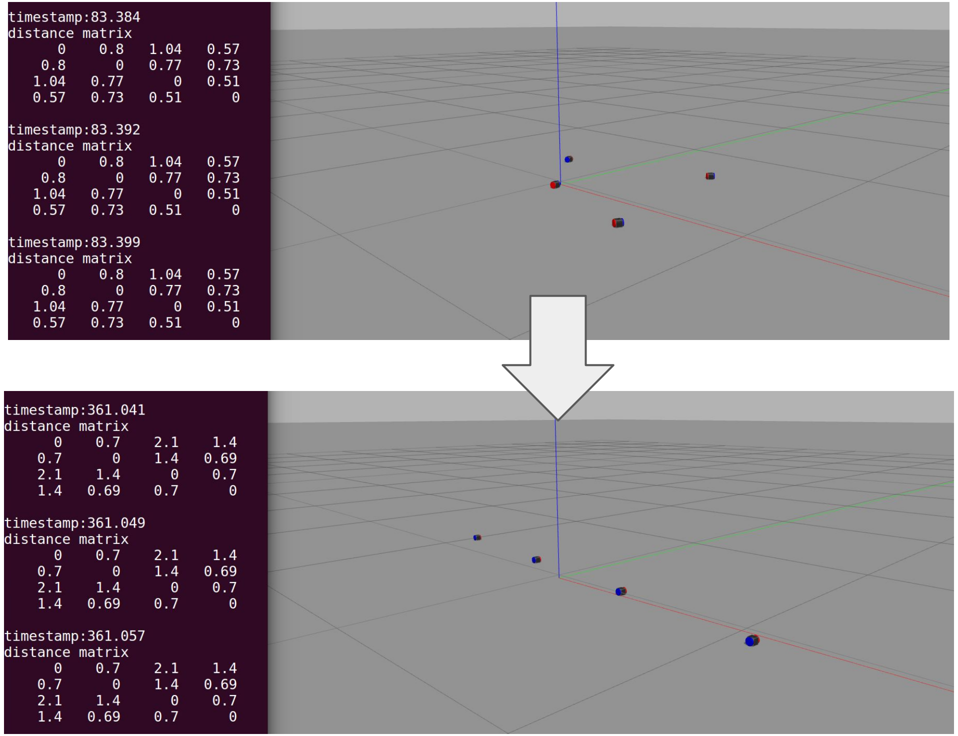
START

- Randomly populated in open world
- Swarm members' actions determined according to local rules

FINISH

- Evenly dispersed
- Microadjustments due to noise from own system and external environment

Motivating Example — Ground Swarm



Invariants from Current Approaches

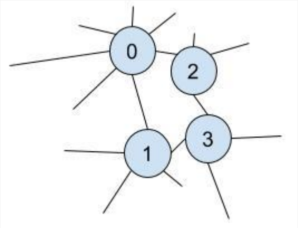
- Cell-wise equivalence
  - dist_matrix[i][j] == dist_matrix[j][i] [1]
- Array relations
  - dist_matrix[i][j] = A[2*i+j] [2]
- Approximate temporals
  - dist_matrix[i][j] <€◯dist_matrix[i][j] where next operator is not strictly enforced [3]

Missing Desired Invariants

- Linear algebraic invariants
  - isSparse == False
- Subswarms
  - Subswarm = {(0,0), (1,1), (2,2), (3,3)}
- Relational approximate temporals
  - norm <€◯norm ± ε

[1] Michael D. Ernst, Jake Cockrell, William G. Griswold, and David Notkin. Dynamically discovering likely program invariants to support program evolution.IEEE Transactions on Software Engineering, 27(2):99–123, February 2001.
[2] Thanhvu Nguyen, Deepak Kapur, Westley Weimer, and Stephanie Forrest. 2014. DIG: A Dynamic Invariant Generator for Polynomial and Array Invariants. ACM Trans. Softw. Eng. Methodol. 23, 4, Article 30 (September 2014), 30 pages. DOI:https://doi.org/10.1145/2556782
[3] Mark Gabel and Zhendong Su. Javert: fully automatic mining of general temporal properties from dynamic traces. In *SIGSOFT FSE*, 2008.

Relational Data Structures

Scalar	Vectors	Matrices	Tensors	Graphs
1	$\begin{bmatrix} 1 & 2 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$	$\begin{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 2 \end{bmatrix} & \begin{bmatrix} 5 & 2 \\ 7 & 2 \end{bmatrix} \end{bmatrix}$	

State of the art

A = null
X < 20
B → A

Arr1 = Arr2
len(Arr1) = 5

Proposed Work

mat isInvertible

tensor isSparse

graph isAcyclic
graph isComplete

Problem Space

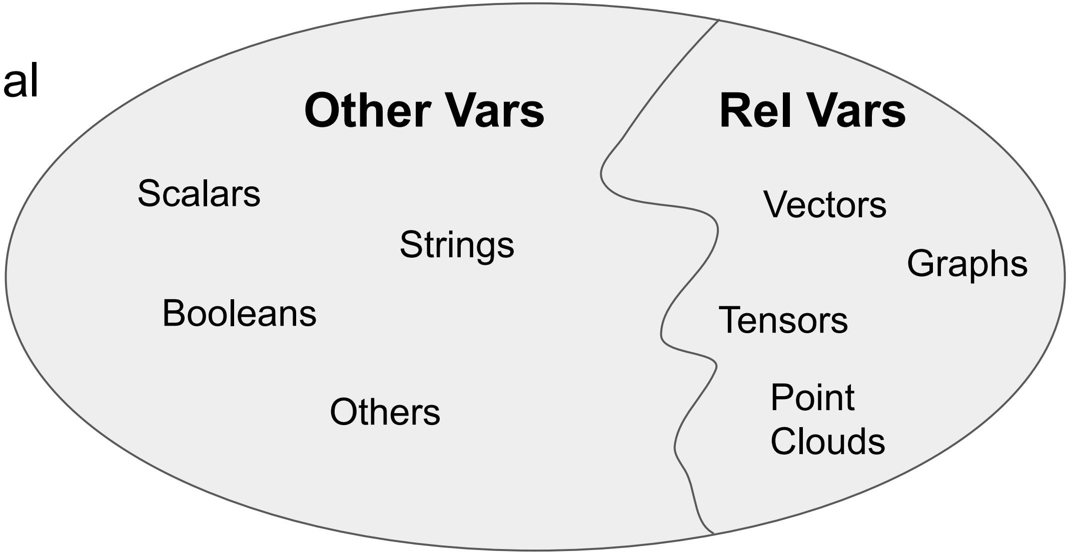
GITHUB REPOSITORY	COMMITS	SLOC	LANG.	ROS?	SIM?	DATA STRUCTS OF INTEREST
yangliu28/swarm_robot_ros_sim	198	3,152	C++	Y	Gazebo	8
xuefengchang/micros_swarm_framework	182	8,990	C++, python	Y	Rviz	18
lucascoelhof/voronoi_hsi	76	1,484	python	Y	Stage	19
or-tal-robotics/mcl_pi	98	706	python	Y	Rviz	10
terna/SLAPP3	145	5,772	python	N	Turtle	4
hanruihua/slave_multirobot	59	6,444	C++	Y	Matplotlib	4
raoshashank/Multi-Robot-Decentralized-Graph-Exploration	83	1,435	C++, python	Y	Gazebo	6
USC-ACTLab/crazyswarm	341	5,912	C++	Y	Cfsim	10
mehdish89/UR5_Cooperative_Transform	582	4,238	C++	Y	Rviz, Gazebo	12
david-alejo/thermal_ws	52	6,469	python	Y	Marble	17
aarow1/cooperative_cable_transport_vision	197	3,442	C++	Y	Rviz	12
awerenne/multi-robot-mapping	75	3,667	python	N	pygame	5
CARMinesDouai/MultiRobotExplorationPackages	104	56,504	C++	Y	Gazebo	4
Koll-Stone/Efficient_sche_cov	47	774	python	N	Matplotlib	7
ThomDietrich/multiUAV-simulation	319	4,867	C++	Y	OMNet++	5
afrl-rq/OpenUxAS	706	464,238	C++, python	Y	Amase	25
mrsd16teamd/MrdRRT	103	1,770	python	Y	matplotlib	6
correlllab/cu-droplet	770	12,008	C++	N	Qt	10
aau-ros/aau_multi_robot	102	15,834	C++	Y	N	26
gondsm/mrgs	363	2,315	C++	Y	N	10
BasJ93/MinorAR_MultiRobot	244	4,088	C++, python	Y	N	2
mzahana/formation	80	1,331	python	Y	Gazebo	8
bramtoula/multi_robot_SLAM_separators	189	4,694	C++	Y	N	5
jimjing/MandM	65	2,831	python	Y	Rviz	8
umass-rbr/multiagent-sas	163	1,257	python	Y	N	7

Table 1: Table of swarm projects.

- Swarm projects collected from Github
- Measurably mature systems
  - ~50 commits
  - ~750 SLOC
  - Had simulation
  - Reference papers
- Data structures of interest
  - Nonzero count for all projects
  - Average 9.92 per project

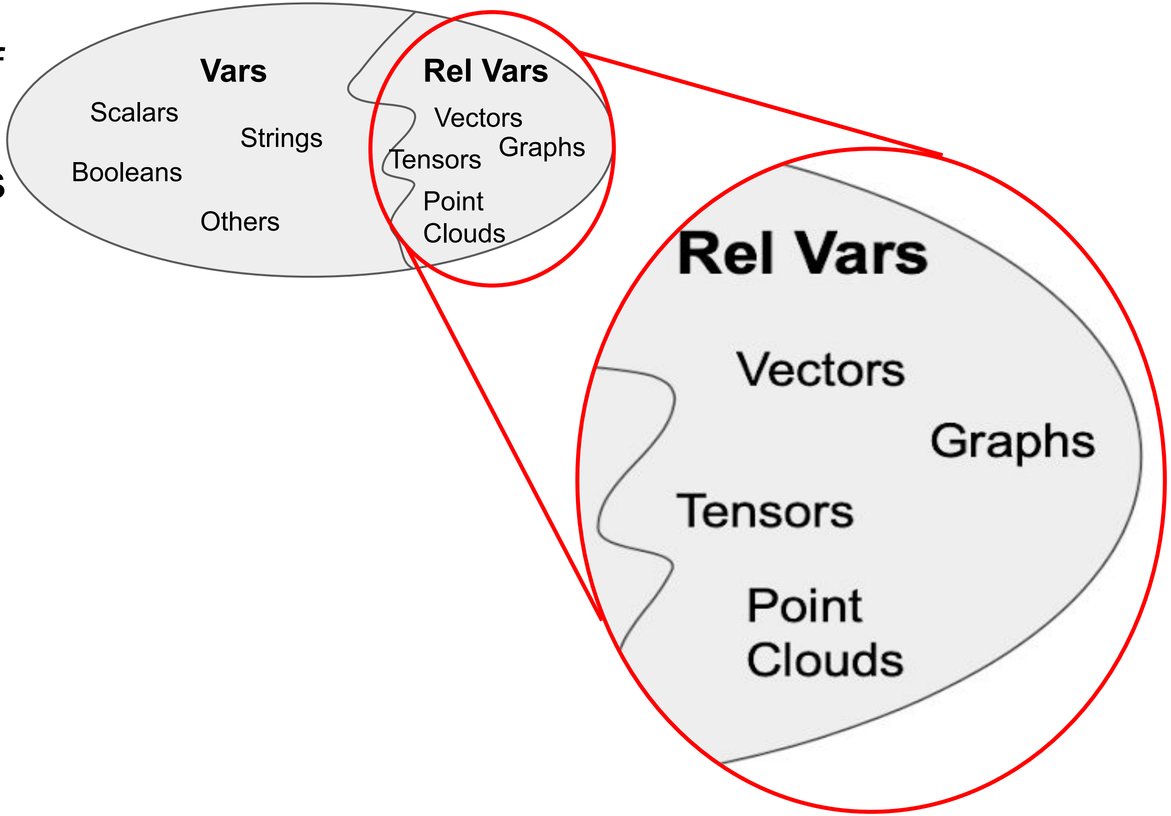
Problem Space

Space of potential variables



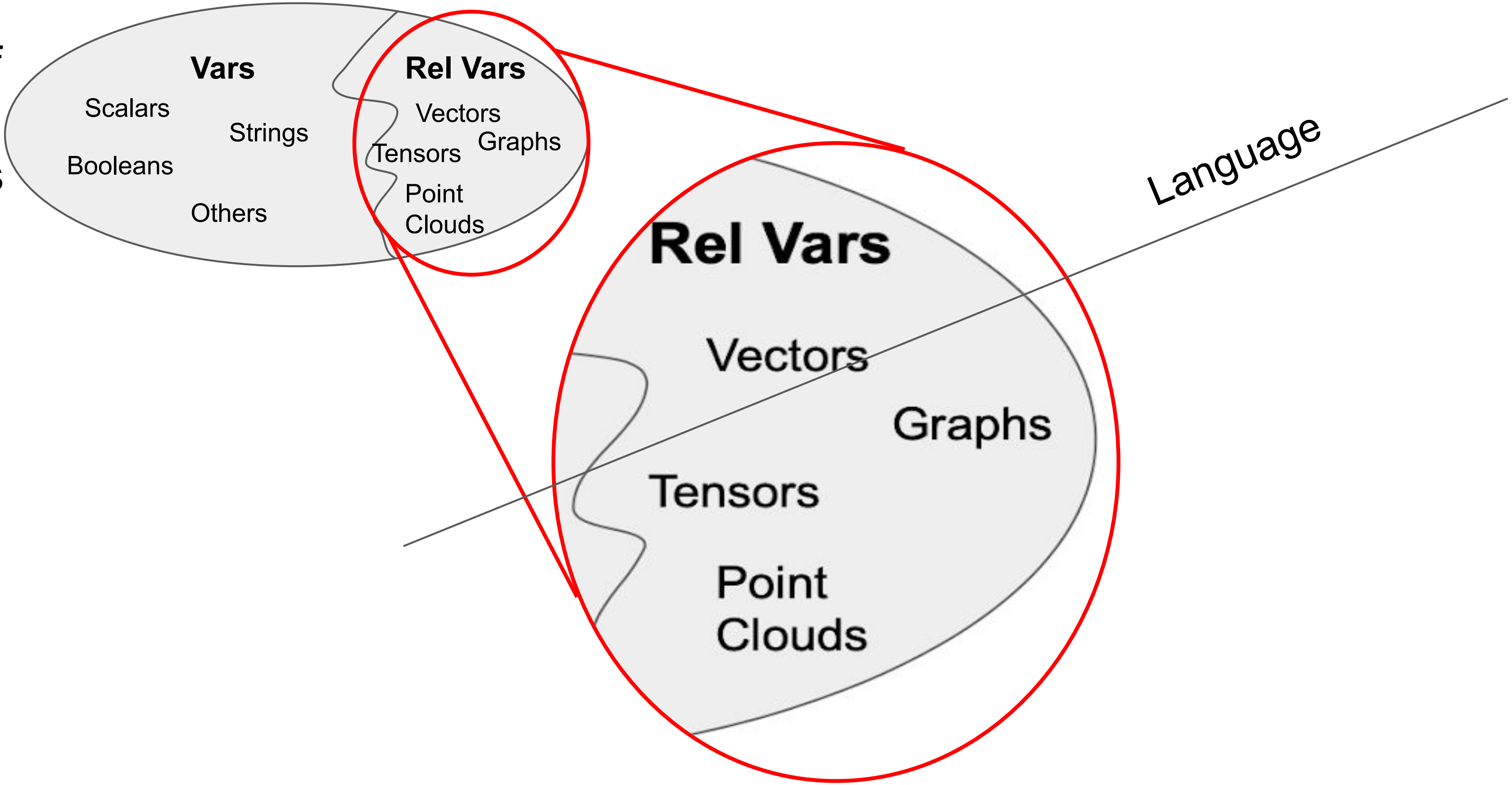
Problem Space

Space of potential variables



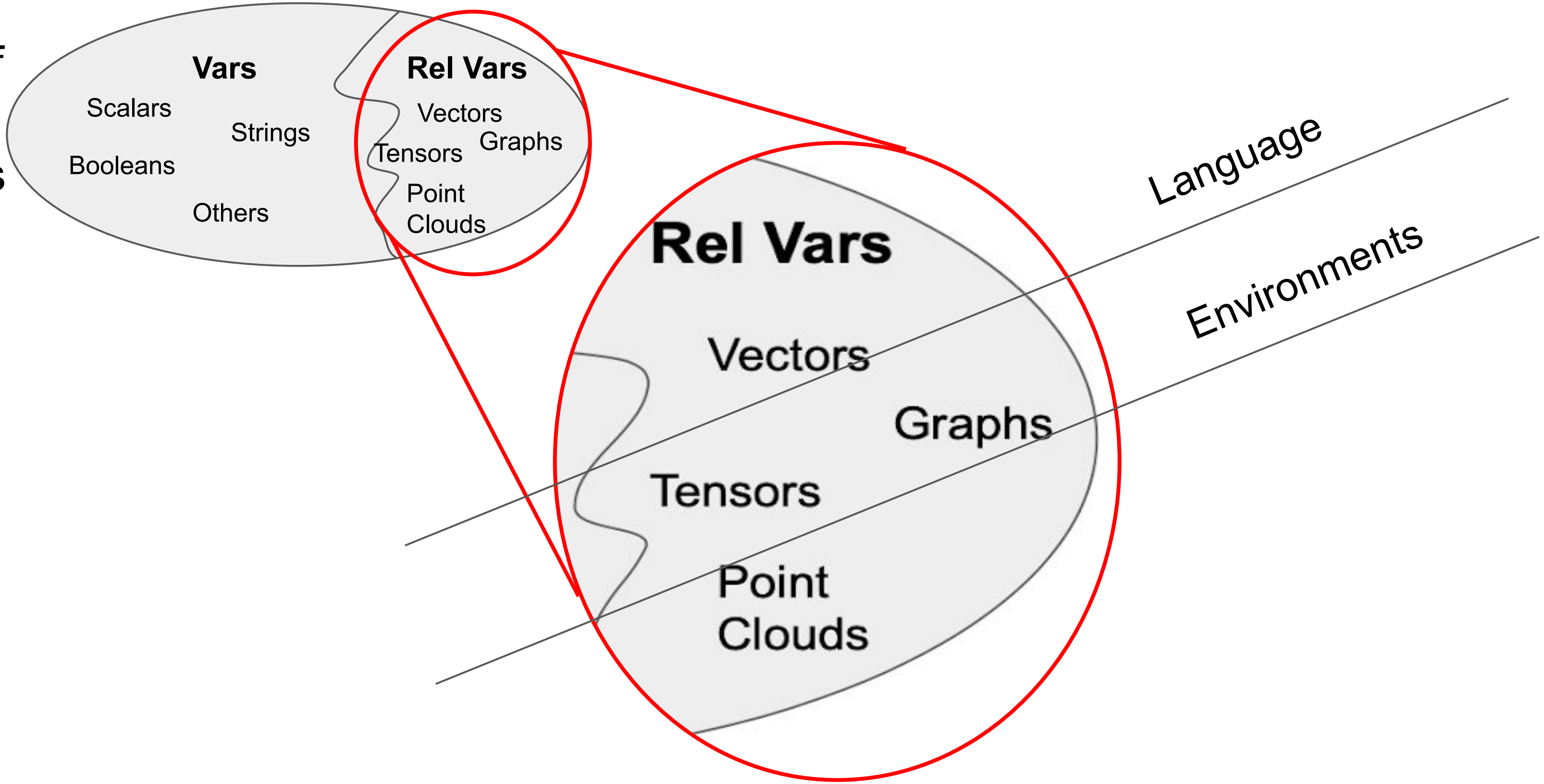
Problem Space

Space of potential variables



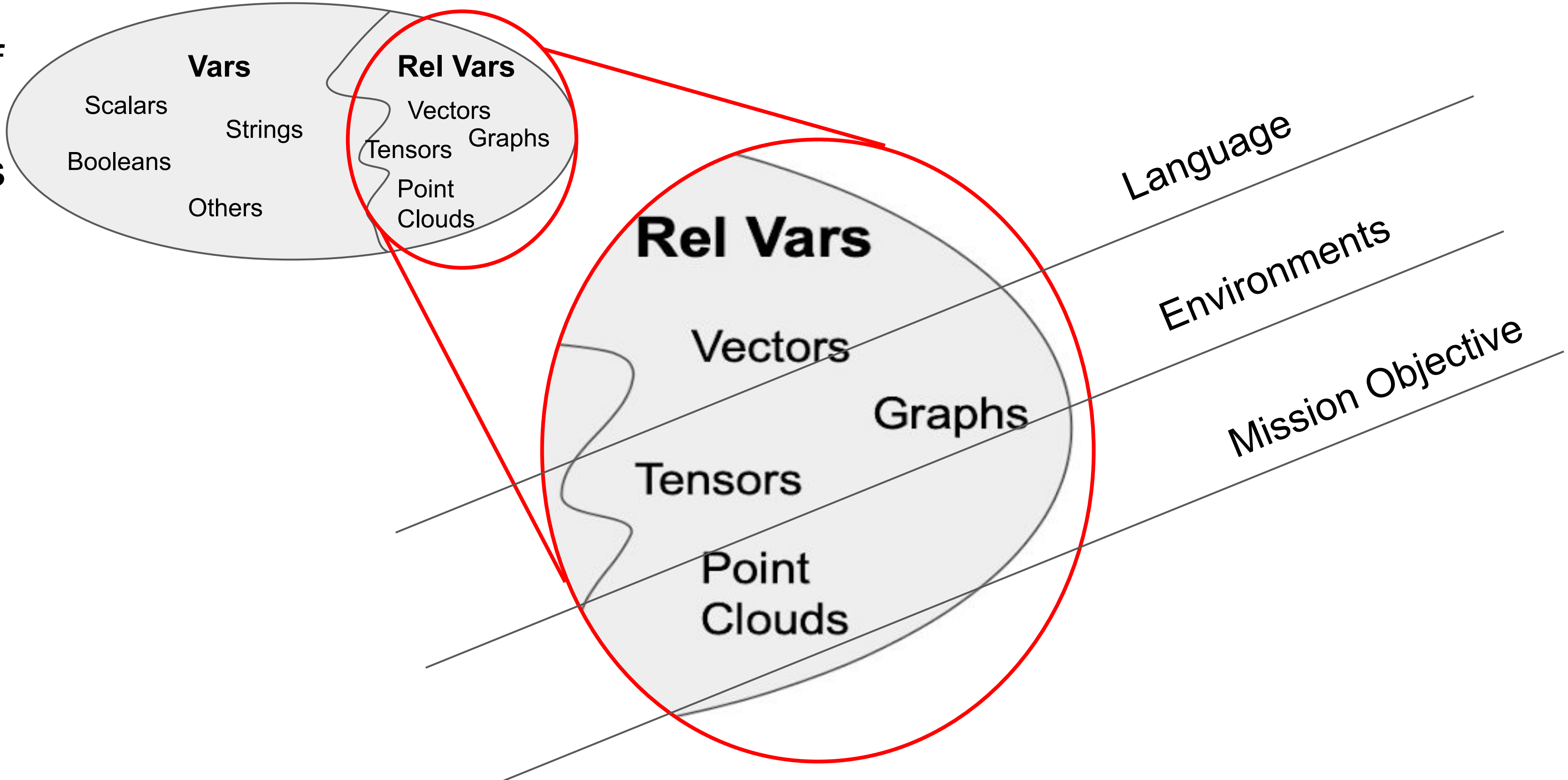
Problem Space

Space of potential variables



Problem Space

Space of potential variables



Approach -- Overview

1. Investigate potentially useful invariant patterns through code analysis.
   a. How are data structures of interest operated upon?
   b. How are they used?
2. Expand upon existing inference techniques according to potentially useful patterns.
3. Optimize inference techniques for the domain of relational data structure invariants.

Approach -- Patterns

1. Investigate potentially useful invariant patterns.

Invariant Type	Pattern	Explanation
Lin. Alg.	isSquare	x-dimension of data structure == y-dimension of data structure. Only applicable to 2D data structures.
Lin. Alg.	isSymmetric	Value at (x,y) equals value at (x,y) of transposed data structure. Applicable to 1D and 2D structures.
Lin. Alg.	isUpperTriangular	Data structure only contains nonzero values above the diagonal. Only applicable to 2D data structures.
Lin. Alg.	isDiagonal	Data structure only contains nonzero values on the diagonal. Only applicable to 2D data structures.
Lin. Alg.	isInvertible	For data structure A, $AA^{-1} = I$. Only applicable to 2D data structures.
Lin. Alg.	isPositive	All values in data structure are nonnegative.
Lin. Alg.	isLinearlyIndependent	Data structure has full rank.
Lin. Alg.	norm	Invariants on the norm values of a data structure. Must be combined with a bound, distribution, or temporal operator.
Lin. Alg.	eigs	Must be combined with a bound, distribution, or temporal operator.
Lin. Alg.	rank	Must be combined with a bound, distribution, or temporal operator.
Lin. Alg.	trace	Must be combined with a bound, distribution, or temporal operator.
Lin. Alg.	isComplex	Data structure contains complex values.
Lin. Alg.	isHermitian	Data structure is a Hermitian matrix. Only applicable to 2D data structures.
Lin. Alg.	determinant	Must be combined with a bound, distribution, or temporal operator.
distribution	A.gaussian(μ, σ)	x values follow a gaussian distribution with mean μ and standard deviation σ.
distribution	max == x	Maximum value of data structure is equal to x.
distribution	min == x	Minimum value of data structure is equal to x.
distribution	mean == x	Mean value of data structure is equal to x.
distribution	median == x	Median value of data structure is equal to x.
distribution	mode == x	Mode value of data structure is equal to x. Mode must occur more than once in individual data structures.
bound	A == B	A is equivalent to B within a user-defined epsilon ball.
bound	$A \leq B$	A is less than or equal to B within a user-defined epsilon ball.
bound	$A \geq B$	A is greater than or equal to B within a user-defined epsilon ball.
bound	$A < B$	A is less than B within a user-defined epsilon ball.
bound	$A > B$	A is greater than B within a user-defined epsilon ball.
subswarms	$\{(x_1, y_1), \ldots (x_n, y_n)\}$	The (x,y) values in the set hold the same values within an epsilon ball at all steps in trace.
temporal	$\bigcirc$ A *op* B	*op* holds for current A and next B at all steps in the trace.
temporal	$\Diamond$ A	Eventually, value A appears in trace. Must be combined with a bound or distribution operator.
temporal	$\Diamond\Box$ A	Eventually, value A always appears in trace. Must be combined with a bound or distribution operator.

Table 5: Currently supported invariant patterns.

- Organized into 5 families
  - Extensible
- Not exhaustive
  - Generally applicable patterns rather than overly specific to systems in problem space
- Chosen for applicability to relational data structures, swarm behavior, or commonly used to describe number sets

Approach -- Patterns

1. Investigate potentially useful invariant patterns.

Linear Algebraic Invariant Patterns

Lin. Alg.	isSquare	x-dimension of data structure == y-dimension of data structure. Only applicable to 2D data structures.
Lin. Alg.	isSymmetric	Value at (x,y) equals value at (x,y) of transposed data structure. Applicable to 1D and 2D structures.
Lin. Alg.	isUpperTriangular	Data structure only contains nonzero values above the diagonal. Only applicable to 2D data structures.
Lin. Alg.	isDiagonal	Data structure only contains nonzero values on the diagonal. Only applicable to 2D data structures.
Lin. Alg.	isInvertible	For data structure A, $AA^{-1} = I$. Only applicable to 2D data structures.
Lin. Alg.	isPositive	All values in data structure are nonnegative.
Lin. Alg.	isLinearlyIndependent	Data structure has full rank.
Lin. Alg.	norm	Invariants on the norm values of a data structure. Must be combined with a bound, distribution, or temporal operator.
Lin. Alg.	eigs	Must be combined with a bound, distribution, or temporal operator.
Lin. Alg.	rank	Must be combined with a bound, distribution, or temporal operator.
Lin. Alg.	trace	Must be combined with a bound, distribution, or temporal operator.
Lin. Alg.	isComplex	Data structure contains complex values.
Lin. Alg.	isHermitian	Data structure is a Hermitian matrix. Only applicable to 2D data structures.
Lin. Alg.	determinant	Must be combined with a bound, distribution, or temporal operator.

- Family of invariants not present in previous work
- Linear algebra meant to characterize high-dimensional data structures
- Used in basic proofs and theorems or came up in code analysis
- Can be combined to point to theorems
    - E.g. Dimension theorem[1]

[1] https://math.mit.edu/~gs/linearalgebra/linearalgebra5_6Great.pdf

Approach -- Patterns

1. Investigate potentially useful invariant patterns.

Linear Algebraic Invariant Patterns

Lin. Alg.	isSquare	x-dimension of data structure == y-dimension of data structure. Only applicable to 2D data structures.
Lin. Alg.	isSymmetric	Value at (x,y) equals value at (x,y) of transposed data structure. Applicable to 1D and 2D structures.
Lin. Alg.	isUpperTriangular	Data structure only contains nonzero values above the diagonal. Only applicable to 2D data structures.
Lin. Alg.	isDiagonal	Data structure only contains nonzero values on the diagonal. Only applicable to 2D data structures.
Lin. Alg.	isInvertible	For data structure A, $AA^{-1} = I$. Only applicable to 2D data structures.
Lin. Alg.	isPositive	All values in data structure are nonnegative.
Lin. Alg.	isLinearlyIndependent	Data structure has full rank.
Lin. Alg.	norm	Invariants on the norm values of a data structure. Must be combined with a bound, distribution, or temporal operator.
Lin. Alg.	eigs	Must be combined with a bound, distribution, or temporal operator.
Lin. Alg.	rank	Must be combined with a bound, distribution, or temporal operator.
Lin. Alg.	trace	Must be combined with a bound, distribution, or temporal operator.
Lin. Alg.	isComplex	Data structure contains complex values.
Lin. Alg.	isHermitian	Data structure is a Hermitian matrix. Only applicable to 2D data structures.
Lin. Alg.	determinant	Must be combined with a bound, distribution, or temporal operator.

- isSymmetric == True
  - Potential optimization point
- isPositive == True
  - Check for bugs
- Norm == 2.828
  - Measure of dispersion
- Determinant == -4.0
  - Measure of average variance

$$\begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix}$$

timestep : 0 timestep : 1 ……. timestep : n timestep : n+1

Approach -- Patterns

1. Investigate potentially useful invariant patterns.

Distributions
Invariant Patterns

distribution	A.gaussian(μ, σ)	x values follow a gaussian distribution with mean μ and standard deviation σ.
distribution	max == x	Maximum value of data structure is equal to x.
distribution	min == x	Minimum value of data structure is equal to x.
distribution	mean == x	Mean value of data structure is equal to x.
distribution	median == x	Median value of data structure is equal to x.
distribution	mode == x	Mode value of data structure is equal to x. Mode must occur more than once in individual data structures.

- Account for noise inherent in robotic systems
- Characterize values occurring in data structures

$$\begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix}$$

timestep : 0 timestep : 1 ……. timestep : n timestep : n+1

- max == 2
- min == 0
- mat.gaussian(1, 1)

Approach -- Patterns

1. Investigate potentially useful invariant patterns.

Bounds
Invariant Patterns

		once in individual data structures.
bound	A == B	A is equivalent to B within a user-defined epsilon ball.
bound	$A \leq B$	A is less than or equal to B within a user-defined epsilon ball.
bound	$A \geq B$	A is greater than or equal to B within a user-defined epsilon ball.
bound	$A < B$	A is less than B within a user-defined epsilon ball.
bound	$A > B$	A is greater than B within a user-defined epsilon ball.

- Account for noise inherent in robotic systems
- Epsilon comparison as defined by user

$\begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix}$ $\begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix}$ $\begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix}$ $\begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix}$

timestep : 0 timestep : 1 ……. timestep : n timestep : n+1

- determinant < norm
- mean == 1
- mean < max
- mean < rank

Approach -- Patterns

1. Investigate potentially useful invariant patterns.

Subswarms
Invariant Patterns

subswarms	$\{(x_1, y_1), \ldots (x_n, y_n)\}$	The (x,y) values in the set hold the same values within an epsilon ball at all steps in trace.

- Parts of the data structure that hold the same (or similar) values at any given timestep
- Reveal interdependent values/cells in data structure

$$\begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix} \quad \cdots\cdots \quad \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1.1 \\ 1.1 & 0 \end{bmatrix}$$

timestep : 0 timestep : 1 ……. timestep : n timestep : n+1

- subswarm ([0,0], [1,1])
- subswarm ([0,1], [1,0])

Approach -- Patterns

1. Investigate potentially useful invariant patterns.

Temporals
Invariant Patterns

temporal	$\bigcirc$ A *op* B	*op* holds for current A and next B at all steps in the trace.
temporal	$\Diamond$ A	Eventually, value A appears in trace. Must be combined with a bound or distribution operator.
temporal	$\Diamond\Box$ A	Eventually, value A always appears in trace. Must be combined with a bound or distribution operator.

- Arrived at through analysis of swarm behavior
  - Next: evolution of swarm behavior over runtime
  - Eventually: reaching a setpoint
  - Eventually always: reaching a stable equilibrium

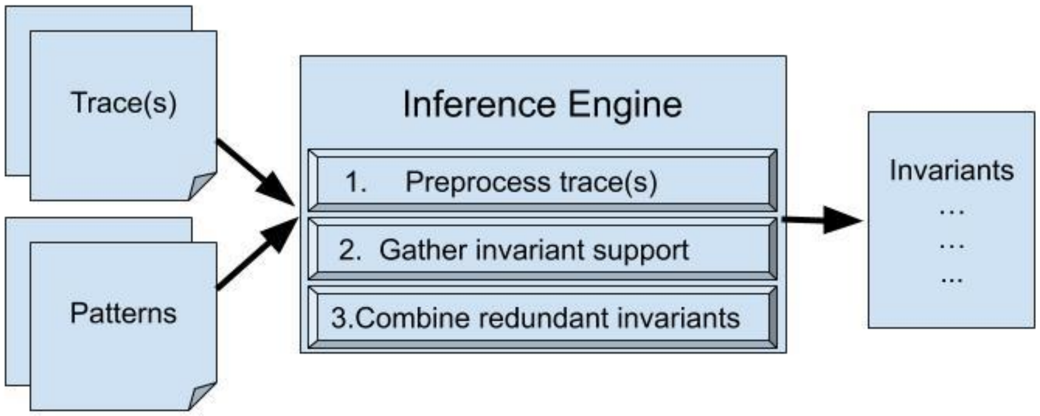
$\begin{bmatrix} 0 & .2 \\ .2 & 0 \end{bmatrix}$ $\begin{bmatrix} 0 & .3 \\ .3 & 0 \end{bmatrix}$ $\begin{bmatrix} 0 & .7 \\ .7 & 0 \end{bmatrix}$ $\begin{bmatrix} 0 & .7 \\ .7 & 0 \end{bmatrix}$

timestep : 0 timestep : 1 ……. timestep : n timestep : n+1

- mean $<\bigcirc$ mean
- $\Diamond\Box$ max == 0.7
- $\Diamond\Box$ norm == 0.9

Approach -- Overview

2. Expand upon existing inference techniques.
3. Optimize inference techniques for the domain of relational data structure invariants.



Approach -- Inference

2. Expand upon existing inference techniques.

Algorithm 1: Inference for linear algebraic operators

```
Input: trace, patterns
Output: results
results = dict();
foreach pattern in patterns do
    foreach record in trace do
        if pattern.eval(record) then
            results.put(pattern, True);
            pattern.count++;
        else
            results.put(pattern, False); break;
        end
    end
    if results.get(pattern) ∧ 1 − 1/pattern.count² <
     0.95 then
        results.put(pattern, False);
    end
end
return results;
```

Algorithm 2: Inference for "eventually always" temporal operator

```
Input: trace, patterns
Output: results
results = dict();
foreach pattern in patterns do
    instantiated = False;
    foreach record in trace do
        highest_possible_conf_interval = 1 -
         1/(trace.length−trace.index(record))²;
        if pattern.eval(record) then
            pattern.count++;
            instantiated = True;
        else if not pattern.eval(record) then
            instantiated = False;
        end
        if highest_possible_conf_interval < 0.95
         then
            results.put(pattern, False); break;
        end
    end
    results.put(pattern, True);
end
return results;
```

Approach -- Inference

2. Expand upon existing inference techniques.

Algorithm 1: Inference for linear algebraic operators

```
Input: trace, patterns
Output: results
results = dict();
foreach pattern in patterns do
    foreach record in trace do
        if pattern.eval(record) then
            results.put(pattern, True);
             pattern.count++;
        else
            results.put(pattern, False); break;
        end
    end
    if results.get(pattern) ∧ 1 − 1/pattern.count² <
     0.95 then
        results.put(pattern, False);
    end
end
return results;
```

- Input: trace, set of patterns
- Output: hashtable containing True/False evaluation for all possible linear algebraic invariants
  - isInvertible == True
- General steps:
  - For each pattern, iterate through trace
  - Pattern holds at that step → increment support for that pattern
  - Pattern does not hold at that step → abort evaluation
  - End of the trace has been reached → Check that support is sufficient

Approach -- Inference

2. Expand upon existing inference techniques.

Algorithm 1: Inference for linear algebraic operators

```
Input: trace, patterns
Output: results
results = dict();
foreach pattern in patterns do
    foreach record in trace do
        if pattern.eval(record) then
            results.put(pattern, True);
            pattern.count++;
        else
            results.put(pattern, False); break;
        end
    end
    if results.get(pattern) ∧ 1 − 1/pattern.count² < 0.95 then
        results.put(pattern, False);
    end
end
return results;
```

predicate: matrix.max <= 0.7

	Trace	Pattern	Confidence
timestep : 0	$\begin{bmatrix} 0 & .2 \\ .2 & 0 \end{bmatrix}$	matrix.max = 0.2	1 - (1 / 1^2) = 0
timestep : 1	$\begin{bmatrix} 0 & .3 \\ .3 & 0 \end{bmatrix}$	matrix.max = 0.3	1 - (1 / 2^2) = 0.75
	....		
timestep : n	$\begin{bmatrix} 0 & .7 \\ .7 & 0 \end{bmatrix}$	matrix.max = 0.7	1 - (1 / n^2) = ...
timestep : n+1	$\begin{bmatrix} 0 & .7 \\ .7 & 0 \end{bmatrix}$	matrix.max = 0.7	1 - (1 / $(n+1)^2$) = ...
	....		

Approach -- Inference

2. Expand upon existing inference techniques.

Algorithm 2: Inference for "eventually always" temporal operator

```
Input: trace, patterns
Output: results
results = dict();
foreach pattern in patterns do
    instantiated = False;
    foreach record in trace do
        highest_possible_conf_interval = 1 - 1/(trace.length−trace.index(record))^2;
        if pattern.eval(record) then
            pattern.count++;
            instantiated = True;
        else if not pattern.eval(record) then
            instantiated = False;
        end
        if highest_possible_conf_interval < 0.95 then
            results.put(pattern, False); break;
        end
    end
    results.put(pattern, True);
end
return results;
```

- Input: trace, set of patterns
- Output: True/False evaluation for all possible eventually always invariants
    - ◇▢ isInvertible
- General steps:
    - For each pattern, iterate through trace
    - If pattern holds at that step, increment support for that pattern
    - Pattern does not hold → is there enough of the trace left for it to receive sufficient support?

Approach -- Inference

2. Expand upon existing inference techniques.

Algorithm 2: Inference for "eventually always" temporal operator

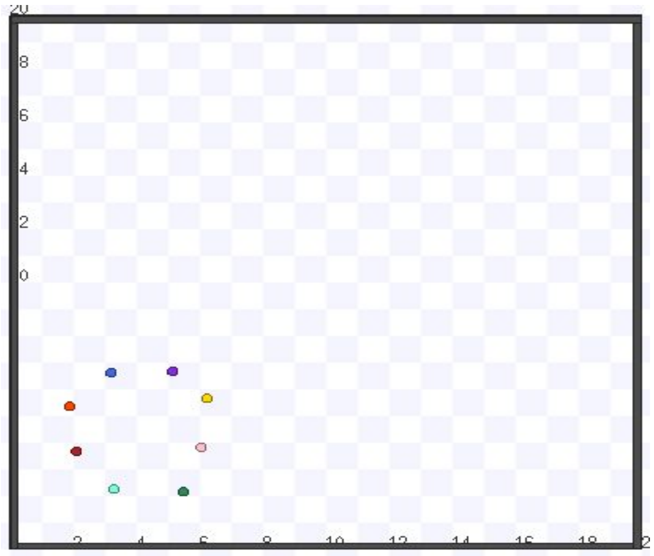
```
Input: trace, patterns
Output: results
results = dict();
foreach pattern in patterns do
    instantiated = False;
    foreach record in trace do
        highest_possible_conf_interval = 1 - 1/(trace.length−trace.index(record))^2;
        if pattern.eval(record) then
            pattern.count++;
            instantiated = True;
        else if not pattern.eval(record) then
            instantiated = False;
        end
        if highest_possible_conf_interval < 0.95 then
            results.put(pattern, False); break;
        end
    end
    results.put(pattern, True);
end
return results;
```

predicate: ◇▢ matrix.max == 0.7

	Trace	Pattern	Confidence
timestep : 0	$\begin{bmatrix} 0 & .2 \\ .2 & 0 \end{bmatrix}$	matrix.max = 0.2	$1 - (1 / 1^2) = 0$
timestep : 1	$\begin{bmatrix} 0 & .3 \\ .3 & 0 \end{bmatrix}$	matrix.max = 0.3	$1 - (1 / 1^2) = 0$
	....		
timestep : n	$\begin{bmatrix} 0 & .7 \\ .7 & 0 \end{bmatrix}$	matrix.max = 0.7	$1 - (1 / 1^2) = 0$
timestep : n+1	$\begin{bmatrix} 0 & .7 \\ .7 & 0 \end{bmatrix}$	matrix.max = 0.7	$1 - (1 / 2^2) = 0.75$
	....		

Approach -- Optimization

3. Optimize inference techniques for the domain of relational data structure invariants.



- Occupancy grid from one of the three case study projects
    - In code: 80 x 80 cell grid
    - Visually: 20 x 20 cell grid
- Want to “zoom out” and deal with an abstracted version of this occupancy grid
- N.b. This is not the only optimization technique that could be applied to this occupancy grid

Research Questions

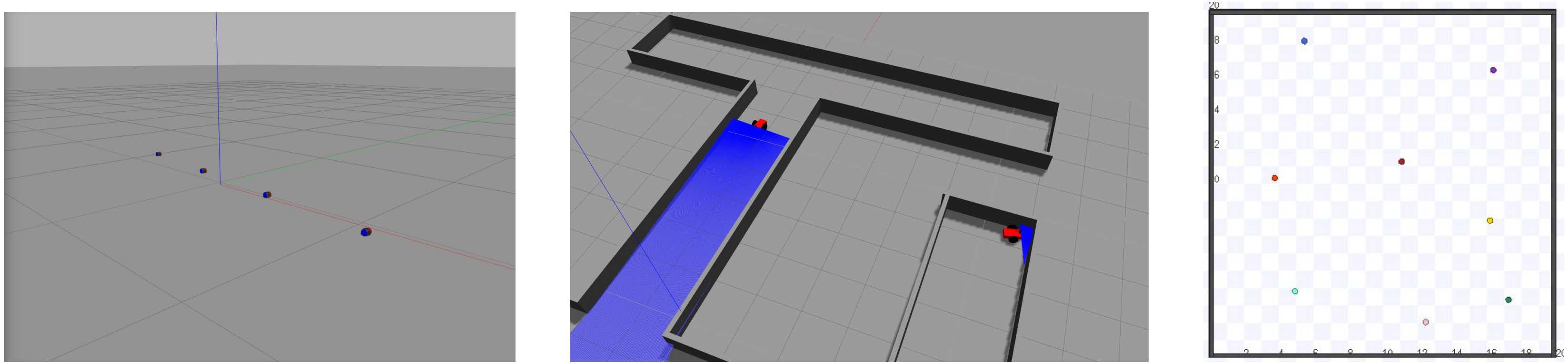
1. Are the posited invariant patterns for relational data structures upheld in practice by robotic systems with high interdependence (swarms)?
2. Can these invariant patterns be used to differentiate between successful and failed behaviors of these swarms?
3. Do the findings for the above two questions suggest the need for further patterns or further expansion of inference techniques?
4. What is the cost associated with generating the posited families of invariants?

Research Questions

1. Are the posited invariant patterns for relational data structures upheld in practice by robotic systems with high interdependence (swarms)?
2. Can these invariant patterns be used to differentiate between successful and failed behaviors of these swarms?
3. Do the findings for the above two questions suggest the need for further patterns or further expansion of inference techniques?
4. What is the cost associated with generating the posited families of invariants?

Study -- Setup

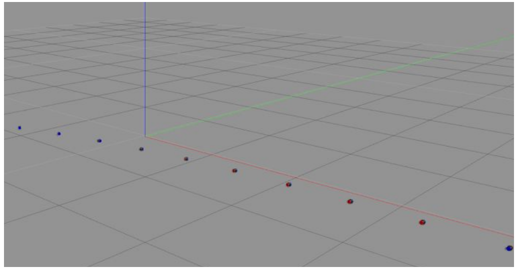
1. Run system in simulation using unperturbed configuration.
2. Run again in adversarially perturbed configuration.
3. Diff invariants generated for both perturbed and unperturbed configurations.

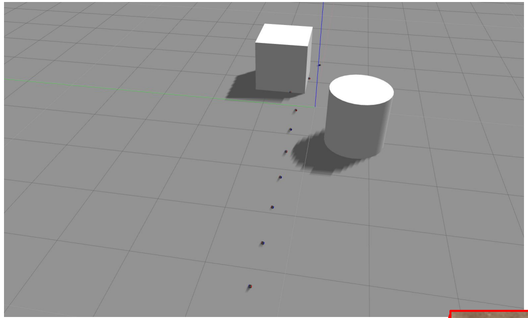


Study -- Results

Invariant Type	Invariant	Explanation	Obstacle Violated?	Mountain Violated?
Shape	isSquare==True	Data structure is a square matrix.	N	N
Lin. Alg.	isSymmetric==True	Data structure is a symmetric matrix.	N	N
Lin. Alg.	isInvertible==True	Data structure is invertible.	N	N
Lin. Alg.	isPositive==True	Data structure is positive, i.e. contains only positive values.	N	N
Lin. Alg.	isLinearlyIndependent==True	Data structure has linearly independent columns.	N	N
Lin. Alg.	7.52<= norm <=28.30	The norm of the data structure falls between these values.	N	Y
Lin. Alg.	norm.gaussian(24.28, 5.31)	Norm values of the data structure follow a gaussian distribution with a mean of 24.28 and standard deviation of 5.31	N	Y
Lin. Alg.	rank == 10 rank.gaussian(10.0, 0.0)	Data structure has rank==10. This makes the gaussian invariant redundant.	N	Y
Lin. Alg.	trace == 0.0 trace.gaussian(0.0, 0.0)	Data structure has trace==0. This makes the gaussian invariant redundant.	N	N
Lin. Alg.	isHermitian==True	Data structure is a Hermitian matrix.	N	N
Lin. Alg.	-56.96<= determinant <=-0.01	Data structure has determinant between these values.	N	Y
bound	0.66<= mean <=2.30	Mean of data structure falls between these values.	N	Y
distribution	mean.gaussian(1.98, 0.42)	Mean of data structure follows a gaussian distribution with a mean of 1.98 and a standard deviation of 0.42.	N	Y
bound	0.655<= median <=2.09	Median value of data structure falls between these values.	N	Y
distribution	median.gaussian(1.78, 0.40)		N	Y
bound	1.33<= maximum <=6.28	Maximum value of data structure falls between these values.	N	Y
distribution	maximum.gaussian(5.44, 1.13)		N	Y
bound	minimum == 0.0 minimum.gaussian(0.0, 0.0)	Minimum value of the data structure is zero. This makes the gaussian invariant redudant.	N	N
subswarm	subswarm: (0,0) (1,1) (2,2) (3,3) (4,4) (5,5) (6,6) (7,7) (8,8) (9,9)]	Subswarm with (x,y) entries found.	N	N
temporal	$\bigcirc$ next: norm@t $\leq$ norm@t+1	Data structure is eventually always not sparse.	N	N
temporal	$\Diamond\Box$ isSparse==False	Data structure is eventually always not sparse.	N	N
temporal	$\Diamond\Box$ norm==28.30	Norm is eventually always 28.30.	N	Y
temporal	$\Diamond\Box$ determinant==-56.96	Determinant is eventually always -56.96.	N	Y
temporal	$\Diamond\Box$ mode==0.69	Mode value in data structure is eventually always 0.69.	Y	Y

Table 2: Invariants generated for distance matrix in yangliu control loop over one successful run.

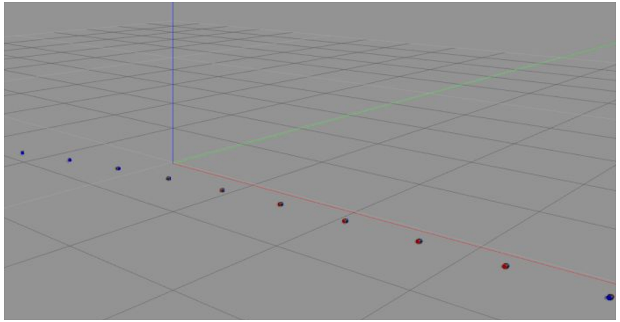




Study -- Results

Invariant Type	Invariant	Explanation	Obstacle Violated?	Mountain Violated?
Shape	isSquare==True	Data structure is a square matrix.	N	N
Lin. Alg.	isSymmetric==True	Data structure is a symmetric matrix.	N	N
Lin. Alg.	isInvertible==True	Data structure is invertible.	N	N
Lin. Alg.	isPositive==True	Data structure is positive, i.e. contains only positive values.	N	N
Lin. Alg.	isLinearlyIndependent==True	Data structure has linearly independent columns.	N	N
Lin. Alg.	7.52<= norm <=28.30	The norm of the data structure falls between these values.	N	Y
Lin. Alg.	norm.gaussian(24.28, 5.31)	Norm values of the data structure follow a gaussian distribution with a mean of 24.28 and standard deviation of 5.31	N	Y
Lin. Alg.	rank == 10 rank.gaussian(10.0, 0.0)	Data structure has rank==10. This makes the gaussian invariant redundant.	N	Y
Lin. Alg.	trace == 0.0 trace.gaussian(0.0, 0.0)	Data structure has trace==0. This makes the gaussian invariant redundant.	N	N
Lin. Alg.	isHermitian==True	Data structure is a Hermitian matrix.	N	N
Lin. Alg.	-56.96<= determinant <=-0.01	Data structure has determinant between these values.	N	Y
bound	0.66<= mean <=2.30	Mean of data structure falls between these values.	N	Y
distribution	mean.gaussian(1.98, 0.42)	Mean of data structure follows a gaussian distribution with a mean of 1.98 and a standard deviation of 0.42.	N	Y
bound	0.655<= median <=2.09	Median value of data structure falls between these values.	N	Y
distribution	median.gaussian(1.78, 0.40)		N	Y
bound	1.33<= maximum <=6.28	Maximum value of data structure falls between these values.	N	Y
distribution	maximum.gaussian(5.44, 1.13)		N	Y
bound	minimum == 0.0 minimum.gaussian(0.0, 0.0)	Minimum value of the data structure is zero. This makes the gaussian invariant redudant.	N	N
subswarm	subswarm: (0,0) (1,1) (2,2) (3,3) (4,4) (5,5) (6,6) (7,7) (8,8) (9,9)]	Subswarm with (x,y) entries found.	N	N
temporal	◯ next: norm@t ≤ norm@t+1	Data structure is eventually always not sparse.	N	N
temporal	◇□ isSparse==False	Data structure is eventually always not sparse.	N	N
temporal	◇□ norm==28.30	Norm is eventually always 28.30.	N	Y
temporal	◇□ determinant==-56.96	Determinant is eventually always -56.96.	N	Y
temporal	◇□ mode==0.69	Mode value in data structure is eventually always 0.69.	Y	Y

Table 2: Invariants generated for distance matrix in yangliu control loop over one successful run.



- Posited invariants are upheld
- Some invariants can distinguish between some successful and failed runs
- Could expand library to better distinguish between minimal failure and catastrophic failure

Research Questions

1. Are the posited invariant patterns for relational data structures upheld in practice by robotic systems with high interdependence (swarms)?

2. Can these invariant patterns be used to differentiate between successful and failed behaviors of these swarms?

3. Do the findings for the above two questions suggest the need for further patterns or further expansion of inference techniques?

4. What is the cost associated with generating the posited families of invariants?

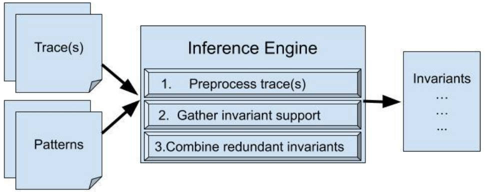
Summary

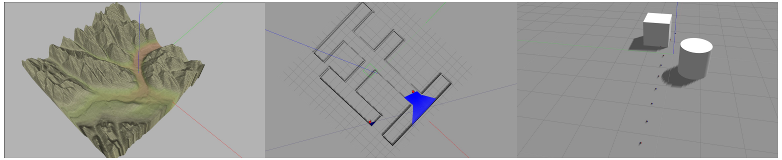
Contributions

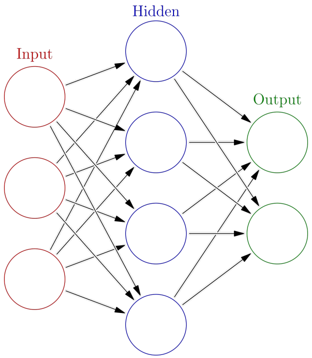
- Expanded inference techniques
- Optimization for larger data structures
- Benchmark of swarm systems

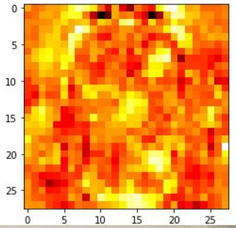
Future Work

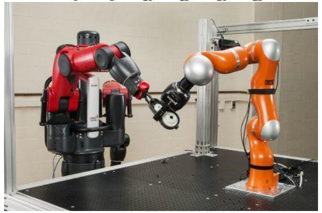
- Apply approach to new systems
    - Heterogeneous cooperative robotic systems
    - Jointed robotic arms
    - Neural networks
- Expanded pattern library
- Approximate pattern matching











Thank You!
Questions

Readings

Invariants — Focus Readings

1. Ivan Beschastnikh, Yuriy Brun, Michael D. Ernst,and Arvind Krishnamurthy. Inferring models of concurrent systems from logs of their behavior with csight. In Proceedings of the 36th International Conference on Software Engineering, ICSE 2014, page 468–479, New York, NY, USA, 2014. Association for Computing Machinery.

2. Christoph Csallner, Nikolai Tillmann, and Yannis Smaragdakis. Dysy: Dynamic symbolic execution for invariant inference. In Proceedings of the 30th International Conference on Software Engineering, ICSE ’08, page 281–290, New York, NY, USA, 2008.Association for Computing Machinery.

3. Michael D. Ernst, Jake Cockrell, William G. Griswold, and David Notkin. Dynamically discovering likely program invariants to support program evolution. IEEE Transactions on Software Engineering, 27(2):99–123, February 2001.

4. Mark Gabel and Zhendong Su. Javert: fully automatic mining of general temporal properties from dynamic traces. In SIGSOFT FSE, 2008.

5. Thanhvu Nguyen, Deepak Kapur, Westley Weimer, and Stephanie Forrest. Dig: A dynamic invariant generator for polynomial and array invariants. ACM Trans. Softw. Eng. Methodol., 23(4):30:1–30:30, September 2014.

6. Jinlin Yang, David Evans, Deepali Bhardwaj, Thirumalesh Bhat, and Manuvir Das. Perracotta: Mining temporal api rules from imperfect traces. In Proceedings of the 28th International Conference on Software Engineering, ICSE ’06, pages 282–291, 2006.

Invariants — Background Readings

1. Tien-Duy B. Le, David Lo, Claire Le Goues, and Lars Grunske. A learning-to-rank based fault localization approach using likely invariants. In Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, page 177–188, New York, NY, USA, 2016. Association for Computing Machinery.

2. Hengle Jiang, Sebastian G. Elbaum, and Carrick Detweiler. Inferring and monitoring invariants in robotic systems. Auton. Robots, 41(4):1027–1046, 2017.

3. Tien-Duy B. Le and David Lo. Deep specification mining. In Proceedings of the 27th ACM SIGSOFT- International Symposium on Software Testing and Analysis, ISSTA 2018, page 106–117, New York, NY,USA, 2018. Association for Computing Machinery.

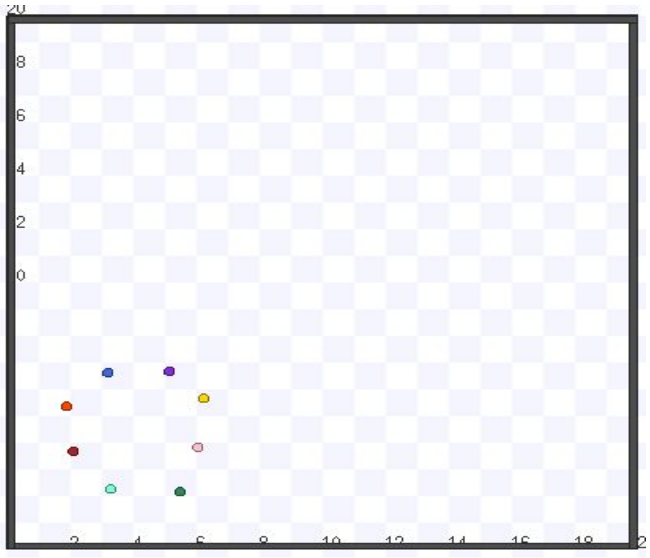
4. L. Grunske. Specification patterns for probabilistic quality properties. In 2008 ACM/IEEE 30th International Conference on Software Engineering, pages 31–40, May 2008.

Supplementary Slides

Approach -- Optimization

3. Optimize inference techniques for the domain of relational data structure invariants.

```
Algorithm 3: Optimization pseudocode
Input: trace, factor
Output: new_trace
size = trace[0].rows × trace[0].columns / factor;
ratio = round(trace[0].rows / trace[0].columns);
rows = size / (factor × ratio);
cols = size / (factor / ratio);
new_trace = [ ];
foreach matrix in trace do
    new_matrix = [rows][cols];
    foreach cell in new_matrix do
        nearest_cells = get_nearest_cells(cell.row,
          cell.col, factor, matrix);
        if matrix.type is int ∨ matrix.type is bool
          then
            cell = max(nearest_cells);
        else
            cell = avg(nearest_cells);
        end
    end
    new_trace.append(new_matrix);
end
return new_trace;
```

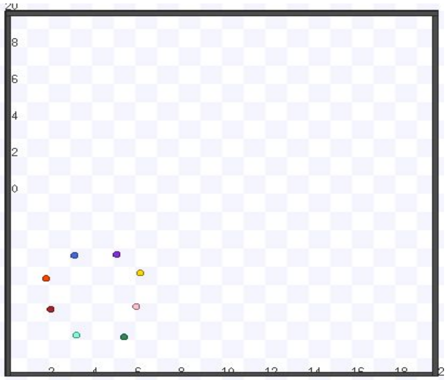


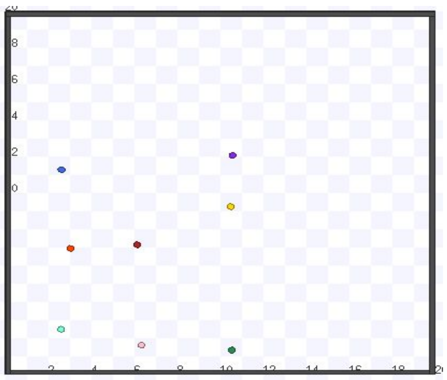
Approach -- Optimization

3. Optimize inference techniques for the domain of relational data structure invariants.

Algorithm 3: Optimization pseudocode

```
Input: trace, factor
Output: new_trace
size = trace[0].rows × trace[0].columns / factor;
ratio = round(trace[0].rows / trace[0].columns);
rows = size / (factor × ratio);
cols = size / (factor / ratio);
new_trace = [ ];
foreach matrix in trace do
    new_matrix = [rows][cols];
    foreach cell in new_matrix do
        nearest_cells = get_nearest_cells(cell.row,
          cell.col, factor, matrix);
        if matrix.type is int ∨ matrix.type is bool
          then
            cell = max(nearest_cells);
        else
            cell = avg(nearest_cells);
        end
    end
    new_trace.append(new_matrix);
end
return new_trace;
```

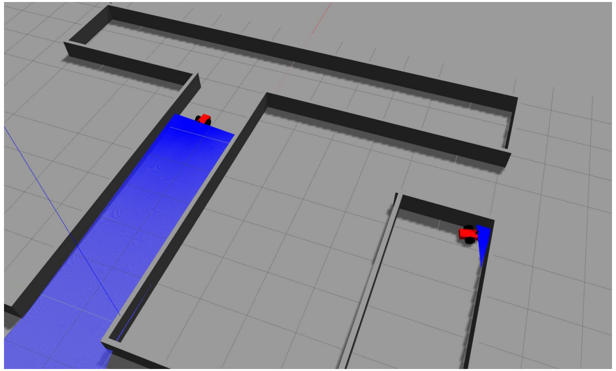


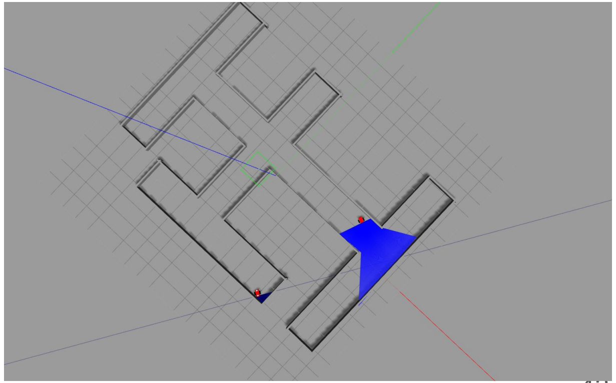


Study -- Results for Case Study #2

Invariant Type	Invariant	Explanation	Violated?
Lin. Alg.	isPositive==True	Data structure is positive, i.e. contains only positive values.	N
Lin. Alg.	isSymmetric==False	Data structure is not a symmetric array.	N
Lin. Alg.	isComplex==False	Data structure contains to complex numbers.	N
Lin. Alg.	isHermitian==False	Data structure is not a Hermitian matrix.	N
Lin. Alg.	isSparse==False	Data structure is not sparse.	N
Lin. Alg.	rank==1	Data structure rank is 1.	N
bound	$99.104 \leq$ norm ≤ 123.333	Norm of the data structure falls between these values	Y
distribution	norm.gaussian(111.480, 7.964)	Norm values of the data structure follow a gaussian distribution with a mean of 111.480 and standard deviation of 7.964	Y
bound	$3.034 \leq$ mean ≤ 3.670	Mean of data structure falls between these values	Y
distribution	mean.gaussian(3.357, 0.205)	Means of the data structure follow a gaussian distribution with a mean of 3.357 and standard deviation of 0.205	Y
bound	$2.505 \leq$ median ≤ 2.638	Medians of data structure fall between these values	Y
distribution	median.gaussian(2.567, 0.040)	Medians of the data structure follow a gaussian distribution with a mean of 3.357 and standard deviation of 0.205	Y
bound	$8.060 \leq$ maximum ≤ 11.210	Maxima of data structure fall between these values	Y
distribution	maximum.gaussian(10.536, 1.105)	Maxima of the data structure follow a gaussian distribution with a mean of 10.536 and standard deviation of 1.105	Y
bound	$0.837 \leq$ minimum ≤ 0.873	Minima of data structure fall between these values	Y
distribution	minimum.gaussian(0.857, 0.006)	Minima of the data structure follow a gaussian distribution with a mean of 0.857 and standard deviation of 0.006	mean N, std Y
temporal	$\bigcirc$ rank@t == rank@t+1	Next rank is equal to the preceding rank.	N

Table 3: Invariants from Multi-Robot-Exploration-Graph project for LaserScan array.

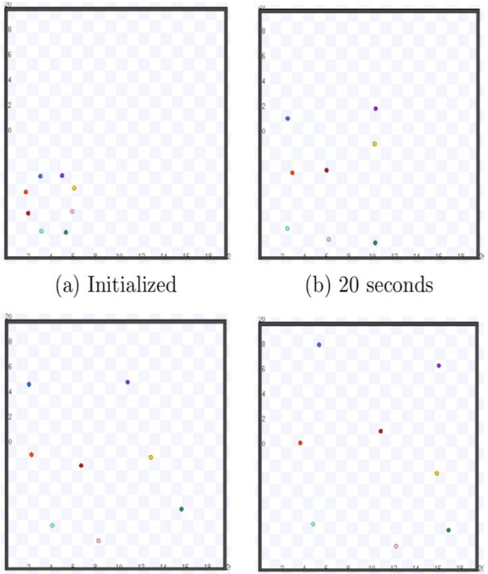




Study -- Results for Case Study #3

Invariant Type	Invariant	Explanation
Lin. Alg.	isSquare==True	Data structure is a square matrix.
Lin. Alg.	isPositive==True	Data structure is positive, i.e. contains only positive values.
Lin. Alg.	isLinearlyIndependent==False	Data structure has linearly independent columns.
Lin. Alg.	isHermitian==False	Data structure is not a Hermitian matrix.
Lin. Alg.	isSparse==True	At least half of values in data structure are zero.
bound	$8 \leq$ rank ≤ 45	Rank of data structure falls between these values.
distribution	rank.gaussian(44.026, 5.923)	Rank of data structure follows a gaussian distribution with a mean of 44.026 and a standard deviation of 5.923.
bound	$1897.367 \leq$ norm ≤ 4602.173	Norm of data structure falls between these values.
distribution	norm.gaussian(4530.994, 432.966)	Norm of data structure follows a gaussian distribution with a mean of 4530.994 and a standard deviation of 432.966.
bound	$5.625 \leq$ mean ≤ 33.094	Mean of data structure falls between these values.
distribution	mean.gaussian(32.371, 4.397)	Mean of data structure follows a gaussian distribution with a mean of 32.371 and a standard deviation of 4.397.
bound	$2.505 \leq$ median ≤ 2.638	Median of the data structure falls between these values.
bound	median == 0.0	Median of data structure is 0.0.
bound	maximum == 100.0	Maximum of data structure is 100.0.
bound	minimum == 0.0	Minimum of data structure is 0.0.
subswarm	(0-1, 0-79), (2-9, 0), (2-9, 1), (2-9, 78), (2-9, 79), (10-16, 0-1), (10, 48-60), (11, 46-62), (12, 45-63), (13, 44-64), (14, 44-65), (15, 43-66), (16, 43-66), (10-16, 78-79)	Subswarm emerged with these (x,y) entries in data structure.

Table 4: Invariants from voronoi_hsi project OccupancyGrid matrix.



(a) Initialized (b) 20 seconds

(c) 1 minute (d) Finished

Study -- Performance

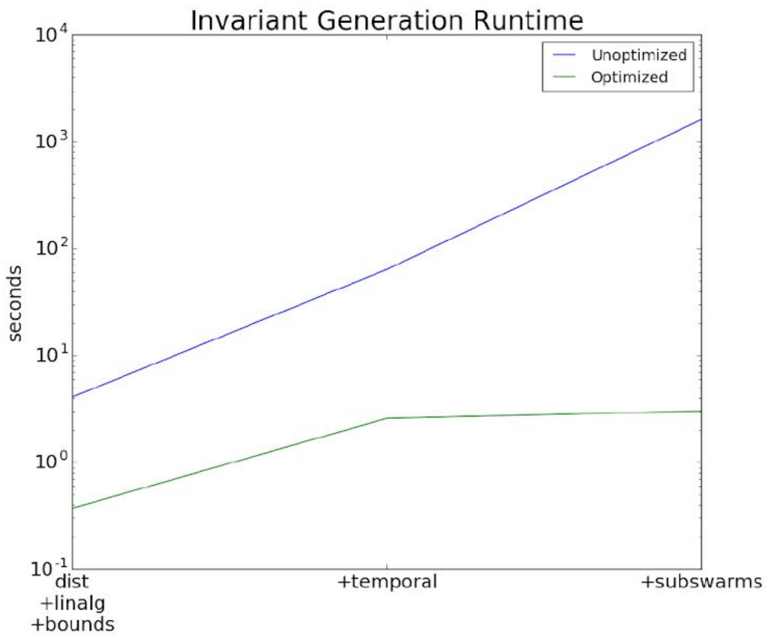


Figure 9: Runtime to compute invariants for 2501 2D data structures derived from a 93-second trace.

- Performance data collected from small trace
    - 2500 instances of 8-cell by 8-cell matrices of type double
- Subswarms invariant inference benefits most from parallelization

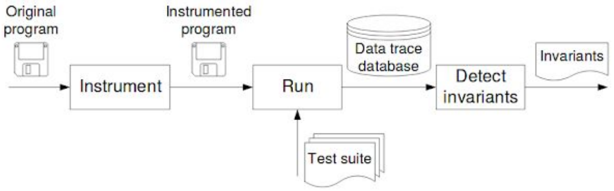
Example -- Neural Network

Weaknesses of Invariants

Inference Processes

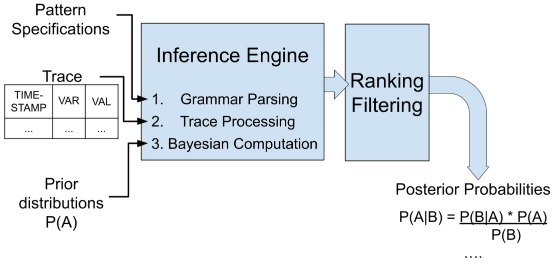
Frequentist

$Event_A == True$



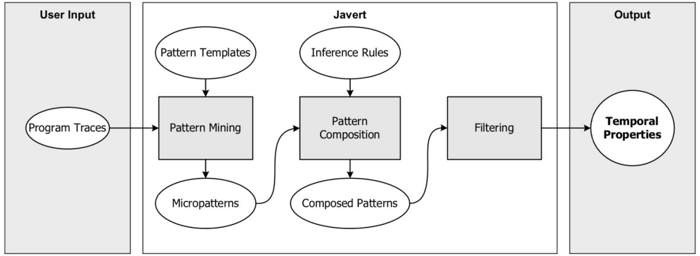
Bayesian

$P(Event_A==True \mid Event_B==True) == 66\%$



Temporal

$Event_A, Event_B, Event_A, Event_B, \ldots == True$



Mark Gabel and Zhendong Su. Javert: fully automatic mining of general temporal properties from dynamic traces. In SIGSOFT FSE, 2008.
Michael D. Ernst, Adam Czeisler, William G. Griswold, and David Notkin. 2000. Quickly detecting relevant program invariants. In *Proceedings of the 22nd international conference on Software engineering* (ICSE '00). ACM, New York, NY, USA, 449-458.

See Slide 15 for further references

Invariants - suggested complement or alternative introduce invariants…

```
public class StackAr{

     private Object [ ] theArray;
     private int        topOfStack;

    public StackAr( int capacity )
    {
        theArray = new Object[ capacity ];
        topOfStack = -1;
    }
     ...
     public Object top( ) {
            if( isEmpty( ) )
                  return null;
            return theArray[ topOfStack ];
     }
     ...
}
```

OBJECT INVARIANTS
this.theArray != null
this.topOfStack >= -1
this.topOfStack <= size(this.theArray[])-1

INVARIANTS AFTER THIS RETURN
return == this.theArray[this.topOfStack]
return == this.theArray[orig(this.topOfStack)]
return == orig(this.theArray[post(this.topOfStack)])
return == orig(this.theArray[this.topOfStack])
this.topOfStack >= 0
return != null

Generated by Daikon from a Stack.Ar Java Class

Approach -- alternative slide to show algorithm

Expand upon existing inference techniques

```
eventuallyAlways(trace, pattern){
        instantiated = False
        foreach record in Trace
                confidence = 1 - 1/(trace.len - trace.index(record))^2
                if pattern.eval(record)
                        instantiated = True
                else
                        if confidence < 0.95 || instantiated==True
                                return False
        return True
}
```

predicate: matrixD.determinant = -56

Trace	Pattern	Confidence	Instantiated	Return
$\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$	matrixD.det = 0			
$\begin{bmatrix} 1 & 2 \\ 5 & 4 \end{bmatrix}$	matrixD.det = -6			
$\begin{bmatrix} 1 & 1 \\ 2 & 4 \end{bmatrix}$	matrixD.det = 2			
$\begin{bmatrix} 1 & 1 \\ 2 & 4 \end{bmatrix}$	matrixD.det = 2			
$\begin{bmatrix} 1 & 1 \\ 2 & 4 \end{bmatrix}$	matrixD.det = 2			

Problem Space