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...

```python
0. def someFunction(x):
1.     y = 2
2.     array1 = [[2 0], [0 0]]
3.     array2 = [x y]
4.     while x < 100:
5.         if(x<=10):
6.             y++
7.             array1[0][0] ++
8.         x++
9.     return y
```

**WHILE LOOP INVARIANTS:**
- \( x < \odot x \)
- \( x \leq 10.0 \rightarrow y > 2 \)
- \( \text{array1}[0][0] == \text{array2}[1] \)
- \( \text{array2}[1] == y \)

**POSTCONDITION INVARIANTS:**
- \( y \leq 12 \)
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

- 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...

https://www.youtube.com/watch?v=ezTayb76x9U
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] \(\leq\) 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 \(\leq\) norm \(\pm\) \(\varepsilon\)

# Relational Data Structures

<table>
<thead>
<tr>
<th>Scalar</th>
<th>Vectors</th>
<th>Matrices</th>
<th>Tensors</th>
<th>Graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image1.png" alt="Vector Example" /></td>
<td><img src="image2.png" alt="Matrix Example" /></td>
<td><img src="image3.png" alt="Tensor Example" /></td>
<td><img src="image4.png" alt="Graph Example" /></td>
<td></td>
</tr>
</tbody>
</table>

### 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

- 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

<table>
<thead>
<tr>
<th>GITHUB REPOSITORY</th>
<th>COMMITS</th>
<th>SLOC</th>
<th>LANG.</th>
<th>ROS?</th>
<th>SIM?</th>
<th>DATA STRUCTS OF INTEREST</th>
</tr>
</thead>
<tbody>
<tr>
<td>yanglin29/swarm_robot_ros_sim</td>
<td>198</td>
<td>3,152</td>
<td>C++</td>
<td>Y</td>
<td>Gazebo</td>
<td></td>
</tr>
<tr>
<td>xuefengchhang/micros_swarm_framework</td>
<td>182</td>
<td>8,990</td>
<td>C++, python</td>
<td>Y</td>
<td>RViz</td>
<td></td>
</tr>
<tr>
<td>lucascoelho/veronoi_lsi</td>
<td>76</td>
<td>1,481</td>
<td>python</td>
<td>Y</td>
<td>Stage</td>
<td></td>
</tr>
<tr>
<td>or-tai-robotics/ncl_pl</td>
<td>98</td>
<td>706</td>
<td>python</td>
<td>Y</td>
<td>RViz</td>
<td></td>
</tr>
<tr>
<td>terna/SLAMPP3</td>
<td>145</td>
<td>5,772</td>
<td>python</td>
<td>N</td>
<td>Turtle</td>
<td></td>
</tr>
<tr>
<td>haurihaha/Slave_multirobot</td>
<td>59</td>
<td>6,444</td>
<td>C++</td>
<td>Y</td>
<td>Matplotlib</td>
<td></td>
</tr>
<tr>
<td>raosheshank/Multi-Robot-Decentralized-Graph-Exploration</td>
<td>83</td>
<td>1,435</td>
<td>C++ , python</td>
<td>Y</td>
<td>Gazebo</td>
<td></td>
</tr>
<tr>
<td>USC-ACTLab/crazyswarm</td>
<td>341</td>
<td>5,912</td>
<td>C++</td>
<td>Y</td>
<td>Csim</td>
<td></td>
</tr>
<tr>
<td>mehdish89/UR3_Cooperative_Transform</td>
<td>582</td>
<td>4,238</td>
<td>C++</td>
<td>Y</td>
<td>RViz, Gazebo</td>
<td></td>
</tr>
<tr>
<td>david-alejo/thermal_ws</td>
<td>52</td>
<td>6,469</td>
<td>python</td>
<td>Y</td>
<td>Marble</td>
<td></td>
</tr>
<tr>
<td>aarow1/cooperative_cable_transport_vision</td>
<td>197</td>
<td>3,442</td>
<td>C++</td>
<td>Y</td>
<td>RViz</td>
<td></td>
</tr>
<tr>
<td>awerence/multi_robot_mapping</td>
<td>75</td>
<td>3,667</td>
<td>python</td>
<td>N</td>
<td>Pygame</td>
<td></td>
</tr>
<tr>
<td>CARMinesDouai/MultiRobotExplorationPackages</td>
<td>104</td>
<td>56,504</td>
<td>C++</td>
<td>Y</td>
<td>Gazebo</td>
<td></td>
</tr>
<tr>
<td>Koll-Stone/Efficient_sche_cw</td>
<td>47</td>
<td>774</td>
<td>python</td>
<td>N</td>
<td>Matplotlib</td>
<td></td>
</tr>
<tr>
<td>ThomDietrich/multiUAV-simulation</td>
<td>319</td>
<td>4,867</td>
<td>C++</td>
<td>Y</td>
<td>OMNet++</td>
<td></td>
</tr>
<tr>
<td>afr4-req/OpenUAS</td>
<td>706</td>
<td>464,238</td>
<td>C++ , python</td>
<td>Y</td>
<td>Amase</td>
<td></td>
</tr>
<tr>
<td>mns1H6eand/MdrRRT</td>
<td>103</td>
<td>1,770</td>
<td>python</td>
<td>Y</td>
<td>Matplotlib</td>
<td></td>
</tr>
<tr>
<td>corellab/cu-droplet</td>
<td>770</td>
<td>12,008</td>
<td>C++</td>
<td>N</td>
<td>Qt</td>
<td></td>
</tr>
<tr>
<td>aau-ros/aau_multi_robot</td>
<td>102</td>
<td>15,834</td>
<td>C++</td>
<td>Y</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>gondsm/mrps</td>
<td>365</td>
<td>2,315</td>
<td>C++</td>
<td>Y</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>BasJ93/MinorAR_MultiRobot</td>
<td>244</td>
<td>4,088</td>
<td>C++, python</td>
<td>Y</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>muzahana/formation</td>
<td>80</td>
<td>1,531</td>
<td>python</td>
<td>Y</td>
<td>Gazebo</td>
<td></td>
</tr>
<tr>
<td>bramtonly/multirobot_SLAM_separators</td>
<td>189</td>
<td>4,694</td>
<td>C++</td>
<td>Y</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>jinjing/MaXmI</td>
<td>65</td>
<td>2,831</td>
<td>python</td>
<td>Y</td>
<td>RViz</td>
<td></td>
</tr>
<tr>
<td>umass-rfr/multiagent-sas</td>
<td>163</td>
<td>1,257</td>
<td>python</td>
<td>Y</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Table of swarm projects.
Problem Space

Space of potential variables

Other Vars
- Scalars
- Strings
- Booleans
- Others

Rel Vars
- Vectors
- Graphs
- Tensors
- Point Clouds

Others
Problem Space

Space of potential variables

- Vars
  - Scalars
  - Strings
  - Booleans
  - Others

- Rel Vars
  - Vectors
  - Tensors
  - Graphs
  - Point
    - Clouds

- Others
  - Scalars
  - Strings
  - Booleans
  - Others
Problem Space

Space of potential variables

- Vars
  - Scalars
  - Strings
  - Booleans
  - Others

- Rel Vars
  - Vectors
  - Tensors
  - Graphs
  - Point Clouds

Rel Vars

- Vectors
- Graphs
- Tensors
- Point Clouds

Language
Problem Space

Space of potential variables

Vars
- Scalars
- Strings
- Booleans
- Others

Rel Vars
- Vectors
- Tensors
- Graphs
- Point Clouds

Rel Vars
- Vectors
- Graphs
- Tensors
- Point Clouds

Language

Environments
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.
1. Investigate potentially useful invariant patterns.

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

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
1. Investigate potentially useful invariant patterns.

### Linear Algebraic Invariant Patterns

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

- 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. Investigate potentially useful invariant patterns.

**Linear Algebraic Invariant Patterns**

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

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

\[
\begin{bmatrix}
0 & 2 \\
2 & 0 \\
\end{bmatrix}
\begin{bmatrix}
0 & 2 \\
2 & 0 \\
\end{bmatrix}
\begin{bmatrix}
0 & 2 \\
2 & 0 \\
\end{bmatrix}
\begin{bmatrix}
0 & 2 \\
2 & 0 \\
\end{bmatrix}
\]

timestep : 0  timestep : 1  .......  timestep : n  timestep : n+1
Approach -- Patterns

1. Investigate potentially useful invariant patterns.

Distributions
Invariant Patterns

<table>
<thead>
<tr>
<th>distribution</th>
<th>A.gaussian($\mu$, $\sigma$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>distribution</td>
<td>max == x</td>
</tr>
<tr>
<td>distribution</td>
<td>min == x</td>
</tr>
<tr>
<td>distribution</td>
<td>mean == x</td>
</tr>
<tr>
<td>distribution</td>
<td>median == x</td>
</tr>
<tr>
<td>distribution</td>
<td>mode == x</td>
</tr>
</tbody>
</table>

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

| timestep : 0 | 0 2 0 |
| timestep : 1 | 0 2 0 |
| timestep : n  | 0 2 0 |
| timestep : n+1 | 0 2 0 |

- max == 2
- min == 0
- mat.gaussian(1, 1)
1. Investigate potentially useful invariant patterns.

<table>
<thead>
<tr>
<th>Bounds</th>
<th>Invariant Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>bound</td>
<td>A == B</td>
</tr>
<tr>
<td>bound</td>
<td>A ≤ B</td>
</tr>
<tr>
<td>bound</td>
<td>A ≥ B</td>
</tr>
<tr>
<td>bound</td>
<td>A &lt; B</td>
</tr>
<tr>
<td>bound</td>
<td>A &gt; B</td>
</tr>
</tbody>
</table>

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

\[
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 2 \\
2 & 0
\end{pmatrix}
\text{timestep : 0} \quad \text{timestep : 1} \quad \ldots \ldots \quad \text{timestep : n} \quad \text{timestep : n+1}
1. Investigate potentially useful invariant patterns.

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

| 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. |
|-----------|------------------------------------------|

\[
\begin{array}{cc}
0 & 2 \\
2 & 0 \\
\end{array} \quad \begin{array}{cc}
0 & 2 \\
2 & 0 \\
\end{array} \quad \begin{array}{cc}
0 & 1 \\
1 & 0 \\
\end{array} \quad \begin{array}{cc}
0 & 1.1 \\
1.1 & 0 \\
\end{array}
\]

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

\[
\begin{array}{c}
timestep : 0 \\
timestep : 1 \\
\ldots \ldots \\
timestep : n \\
timestep : n+1 \\
\end{array}
\]
Approach -- Patterns

1. Investigate potentially useful invariant patterns.

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

<table>
<thead>
<tr>
<th>temporal</th>
<th>□ A op B</th>
<th>op holds for current A and next B at all steps in the trace.</th>
</tr>
</thead>
<tbody>
<tr>
<td>temporal</td>
<td>◊ A</td>
<td>Eventually, value A appears in trace. Must be combined with a bound or distribution operator.</td>
</tr>
<tr>
<td>temporal</td>
<td>◊□ A</td>
<td>Eventually, value A always appears in trace. Must be combined with a bound or distribution operator.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>timestep : 0</th>
<th>timestep : 1</th>
<th>timestep : n</th>
<th>timestep : n+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 .2 .2</td>
<td>0 .3 .3</td>
<td>0 .7 0</td>
<td>0 .7 0</td>
</tr>
</tbody>
</table>

- mean $\leq$ mean
- ◊ max == 0.7
- ◊ norm == 0.9
2. Expand upon existing inference techniques.
3. Optimize inference techniques for the domain of relational data structure invariants.
2. Expand upon existing inference techniques.

---

**Algorithm 1:** Inference for linear algebraic operators

**Input:** trace, patterns

**Output:** results

```java
1 results = dict();
2 foreach pattern in patterns do
3     foreach record in trace do
4         if pattern.eval(record) then
5             results.put(pattern, True);
6             pattern.count++;
7         else
8             results.put(pattern, False); break;
9     end
10    if results.get(pattern) ∧ \frac{1}{pattern\textunderscore\text{count}^2} < 0.95 then
11       results.put(pattern, False);
12 end
13 return results;
```

---

**Algorithm 2:** Inference for “eventually always” temporal operator

**Input:** trace, patterns

**Output:** results

```java
1 results = dict();
2 foreach pattern in patterns do
3     instantiated = False;
4     foreach record in trace do
5         highest\textunderscore\text{possible\textunderscore}\text{conf\textunderscore}interval = 1 - \frac{1}{(trace\textunderscore\text{length} - trace\textunderscore\text{index} [record])^2};
6         if pattern.eval(record) then
7             pattern.count++;
8             instantiated = True;
9         else if not pattern.eval(record) then
10            instantiated = False;
11        end
12    if highest\textunderscore\text{possible\textunderscore}\text{conf\textunderscore}interval < 0.95 then
13        results.put(pattern, False); break;
14    end
15 end
16 results.put(pattern, True);
17 return results;
```
2. Expand upon existing inference techniques.

- **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
2. Expand upon existing inference techniques.

**Algorithm 1: Inference for linear algebraic operators**

<table>
<thead>
<tr>
<th>Trace</th>
<th>Pattern</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestep : 0</td>
<td>0 .2</td>
<td>matrix.max = 0.2</td>
</tr>
<tr>
<td></td>
<td>.2 0</td>
<td>matrix.max = 0.2</td>
</tr>
<tr>
<td>timestep : 1</td>
<td>0 .3</td>
<td>matrix.max = 0.3</td>
</tr>
<tr>
<td></td>
<td>.3 0</td>
<td></td>
</tr>
<tr>
<td>......</td>
<td></td>
<td></td>
</tr>
<tr>
<td>timestep : n</td>
<td>0 .7</td>
<td>matrix.max = 0.7</td>
</tr>
<tr>
<td></td>
<td>.7 0</td>
<td></td>
</tr>
<tr>
<td>......</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2. Expand upon existing inference techniques.

- 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?

Algorithm 2: Inference for “eventually always” temporal operator

```
Input: trace, patterns
Output: results

1 results = dict();
2 foreach pattern in patterns do
3     instantiated = False;
4     foreach record in trace do
5         highest_possible_conf_interval = 1 -
6         (trace.length - trace.index(record))²;
7         if pattern.eval(record) then
8             pattern.count++;  
9             instantiated = True;
10        else if not pattern.eval(record) then
11            instantiated = False;
12        end
13        if highest_possible_conf_interval < 0.95 then
14            results.put(pattern, False); break;
15        end
16     end
17     results.put(pattern, True);
18 return results;
```
Approach -- Inference

2. Expand upon existing inference techniques.

```
Algorithm 2: Inference for “eventually always”
temporal operator

Input: trace, patterns
Output: results
1 results = dict();
2 foreach pattern in patterns do
3      instantiated = False;
4      foreach record in trace do
5          highest_possible_conf_interval = 1 -
6          (trace.length - trace.index(record))²;
7          if pattern.eval(record) then
8              pattern.count++;    
9              instantiated = True;
10         else if not pattern.eval(record) then
11             instantiated = False;
12         end
13     if highest_possible_conf_interval < 0.95
14        then
15           results.put(pattern, False); break;
16     end
17     results.put(pattern, True);
18 end
19 return results;
```

Predicate: $\Diamond □$ matrix.max == 0.7

<table>
<thead>
<tr>
<th>Trace</th>
<th>Pattern</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestep: 0</td>
<td>0 .2</td>
<td>matrix.max = 0.2</td>
</tr>
<tr>
<td>timestep: 1</td>
<td>0 .3</td>
<td>matrix.max = 0.3</td>
</tr>
<tr>
<td>timestep: n</td>
<td>0 .7</td>
<td>matrix.max = 0.7</td>
</tr>
<tr>
<td>timestep: n+1</td>
<td>0 .7</td>
<td>matrix.max = 0.7</td>
</tr>
</tbody>
</table>

......
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?
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

<table>
<thead>
<tr>
<th>Invariant Type</th>
<th>Invariant</th>
<th>Explanation</th>
<th>Obstacle Violated?</th>
<th>Mountain Violated?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shape</td>
<td>isSquare==True</td>
<td>Data structure is a square matrix.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>isSymmetric==True</td>
<td>Data structure is a symmetric matrix.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>isInvertible==True</td>
<td>Data structure is invertible.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>isPositive==True</td>
<td>Data structure is positive, i.e. contains only positive values.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>isLinearlyIndependent==True</td>
<td>Data structure has linearly independent columns.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>7.52&lt;= norm &lt;=28.30</td>
<td>The norm of the data structure falls between these values.</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>norm.gaussian(24.28, 5.31)</td>
<td>Norm values of the data structure follow a gaussian distribution with a mean of 24.28 and standard deviation of 5.31</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>rank == 10</td>
<td>Data structure has rank==10. This makes the gaussian invariant redundant.</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>rank.gaussian(10.0, 0.0)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>trace == 0.0</td>
<td>Data structure has trace==0. This makes the gaussian invariant redundant.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>trace.gaussian(0.0, 0.0)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>isHermitian==True</td>
<td>Data structure is a Hermitian matrix.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>-56.96&lt;= determinant &lt;=0.01</td>
<td>Data structure has determinant between these values.</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>bound</td>
<td>0.66&lt;= mean &lt;=2.30</td>
<td>Mean of data structure falls between these values.</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>distribution</td>
<td>mean.gaussian(1.98, 0.42)</td>
<td>Mean of data structure follows a gaussian distribution with a mean of 1.98 and a standard deviation of 0.42.</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>bound</td>
<td>0.655&lt;= median &lt;=2.09</td>
<td>Median value of data structure falls between these values.</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>distribution</td>
<td>median.gaussian(1.78, 0.40)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bound</td>
<td>1.33&lt;= maximum &lt;=6.28</td>
<td>Maximum value of data structure falls between these values.</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>distribution</td>
<td>maximum.gaussian(5.44, 1.13)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bound</td>
<td>minimum == 0.0</td>
<td>Minimum value of the data structure is zero.</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>minimum</td>
<td>minimum.gaussian(0.0, 0.0)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>subswarm</td>
<td>subswarm: (0.0) (1.1) (2.2) (3.3)</td>
<td>Subswarm with (x,y) entries found.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>subswarm</td>
<td>(4.4) (5.5) (6.6) (7.7) (8.8) (9.9)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>temporal</td>
<td>⊕ next norm@t ≤ norm@t+1</td>
<td>Data structure is eventually always not sparse.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>temporal</td>
<td>⊕ isSparse==False</td>
<td>Data structure is eventually always not sparse.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>temporal</td>
<td>⊕ norm==28.30</td>
<td>Norm is eventually always 28.30.</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>temporal</td>
<td>⊕ determinant==56.96</td>
<td>Determinant is eventually always 56.96.</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>temporal</td>
<td>⊕ node==0.69</td>
<td>Node value in data structure is eventually always 0.69.</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Table 2: Invariants generated for distance matrix in yangu in control loop over one successful run.
Study -- Results

<table>
<thead>
<tr>
<th>Invariant Type</th>
<th>Invariant</th>
<th>Explanation</th>
<th>Obscure Violated?</th>
<th>Mountain Violated?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shape</td>
<td>isSquare==True</td>
<td>Data structure is a square matrix.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>isSymmetric==True</td>
<td>Data structure is a symmetric matrix.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>isInvertible==True</td>
<td>Data structure is invertible.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>isPositive==True</td>
<td>Data structure is positive, i.e. contains only positive values.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>isLinearlyIndependent==True</td>
<td>Data structure has linearly independent columns.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>7.52&lt;= norm &lt;=28.30</td>
<td>The norm of the data structure falls between these values.</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>norm.gaussian(24.28, 5.31)</td>
<td>Norm values of the data structure follow a Gaussian distribution with a mean of 24.28 and standard deviation of 5.31.</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>rank == 10</td>
<td>Data structure has rank==10. This makes the Gaussian invariant redundant.</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>rank.gaussian(10.0, 0.0)</td>
<td>Rank value is Gaussian distributed.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>trace == 0.0</td>
<td>Data structure has trace==0. This makes the Gaussian invariant redundant.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>isHermitian==True</td>
<td>Data structure is a Hermitian matrix.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>-56.96&lt;= determinant &lt;=0.01</td>
<td>Data structure has determinant between these values.</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>bound</td>
<td>0.66&lt;= mean &lt;=2.30</td>
<td>Mean of data structure falls between these values.</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>distribution</td>
<td>mean.gaussian(1.98, 0.42)</td>
<td>Mean of data structure follows a Gaussian distribution with a mean of 1.98 and a standard deviation of 0.42.</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>bound</td>
<td>0.65&lt;= median &lt;=2.09</td>
<td>Median value of data structure falls between these values.</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>distribution</td>
<td>median.gaussian(1.78, 0.30)</td>
<td>Median value of data structure is Gaussian distributed.</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>bound</td>
<td>1.33&lt;= maximum &lt;=6.28</td>
<td>Maximum value of data structure falls between these values.</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>distribution</td>
<td>maximum.gaussian(5.44, 1.13)</td>
<td>Maximum value of data structure is Gaussian distributed.</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>bound</td>
<td>minimum == 0.0</td>
<td>Minimum value of data structure is zero. This makes the Gaussian invariant redundant.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>subswarm</td>
<td>subswarm: (0.0) (1.1) (2.2) (3.3) (4.4) (5.5) (6.6) (7.7) (8.8) (9.9)</td>
<td>Subswarm with (x,y) entries found.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>temporal</td>
<td>next: norm@t &lt; norm@t+1</td>
<td>Data structure is eventually always not sparse.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>temporal</td>
<td>⊕ i Source==False</td>
<td>Data structure is eventually always not sparse.</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>temporal</td>
<td>⊕· norm==0.28.30</td>
<td>Norm is eventually always 28.30.</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>temporal</td>
<td>⊕· determinant==56.96</td>
<td>Determinant is eventually always 56.96.</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>temporal</td>
<td>⊕· mode==0.69</td>
<td>Mode value in data structure is eventually always 0.69.</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

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 — Background Readings


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 = \frac{\text{size}}{\text{factor} \times \text{ratio}};
cols = \frac{\text{size}}{\text{factor} \times \text{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
end
new_trace.append(new_matrix);
return new_trace;
```
3. Optimize inference techniques for the domain of relational data structure invariants.

**Algorithm 3: Optimization pseudocode**

```
Input: trace, factor
Output: new_trace
1. size = trace[0].rows × trace[0].columns / factor;
2. ratio = round((trace[0].rows / trace[0].columns);
3. rows = \(\frac{\text{size}}{(\text{factor} \times \text{ratio})}\);
4. cols = \(\frac{\text{size}}{(\text{factor} / \text{ratio})}\);
5. new_trace = [];
6. foreach matrix in trace do
7.    new_matrix = [rows][cols];
8.    foreach cell in new_matrix do
9.        nearest_cells = get_nearest_cells(cell.row, cell.col, factor, matrix);
10.       if matrix.type is int ∨ matrix.type is bool
11.         then
12.             cell = max(nearest_cells);
13.         else
14.             cell = avg(nearest_cells);
15.         end
16.     end
17. new.trace.append(new_matrix);
18. return new_trace;
```
### Study -- Results for Case Study #2

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

Table 3: Invariants from Multi-Robot-Exploration-Graph project for LaserScan array.
# Study -- Results for Case Study #3

<table>
<thead>
<tr>
<th>Invariant Type</th>
<th>Invariant</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lin. Alg.</td>
<td>isSquare==True</td>
<td>Data structure is a square matrix.</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>isPositive==True</td>
<td>Data structure is positive, i.e. contains only positive values.</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>isLinearlyIndependent==False</td>
<td>Data structure has linearly independent columns.</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>isHermitian==False</td>
<td>Data structure is not a Hermitian matrix.</td>
</tr>
<tr>
<td>Lin. Alg.</td>
<td>isSparse==True</td>
<td>At least half of values in data structure are zero.</td>
</tr>
<tr>
<td>bound</td>
<td>8 \leq \text{rank} \leq 45</td>
<td>Rank of data structure falls between these values.</td>
</tr>
<tr>
<td>distribution</td>
<td>rank.gaussian(44.026, 5.923)</td>
<td>Rank of data structure follows a gaussian distribution with a mean of 44.026 and a standard deviation of 5.923.</td>
</tr>
<tr>
<td>bound</td>
<td>1897.367 \leq \text{norm} \leq 4602.173</td>
<td>Norm of data structure falls between these values.</td>
</tr>
<tr>
<td>distribution</td>
<td>norm.gaussian(4530.994, 432.966)</td>
<td>Norm of data structure follows a gaussian distribution with a mean of 4530.994 and a standard deviation of 432.966.</td>
</tr>
<tr>
<td>bound</td>
<td>5.625 \leq \text{mean} \leq 33.094</td>
<td>Mean of data structure falls between these values.</td>
</tr>
<tr>
<td>distribution</td>
<td>mean.gaussian(32.371, 4.397)</td>
<td>Mean of data structure follows a gaussian distribution with a mean of 32.371 and a standard deviation of 4.397.</td>
</tr>
<tr>
<td>bound</td>
<td>2.505 \leq \text{median} \leq 2.638</td>
<td>Median of the data structure falls between these values.</td>
</tr>
<tr>
<td>bound</td>
<td>median == 0.0</td>
<td>Median of data structure is 0.0.</td>
</tr>
<tr>
<td>bound</td>
<td>maximum == 100.0</td>
<td>Maximum of data structure is 100.0.</td>
</tr>
<tr>
<td>bound</td>
<td>minimum == 0.0</td>
<td>Minimum of data structure is 0.0.</td>
</tr>
<tr>
<td>subswarm</td>
<td>(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)</td>
<td>Subswarm emerged with these (x,y) entries in data structure.</td>
</tr>
</tbody>
</table>

Table 4: Invariants from voronoi_hsi project OccupancyGrid matrix.
Study -- Performance

- 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

Figure 9: Runtime to compute invariants for 2501 2D data structures derived from a 93-second trace.
Example -- Neural Network
Weaknesses of Invariants
Inference Processes

Frequentist

Event\textsubscript{A} == True

Bayesian

P(\text{Event}\textsubscript{A} == \text{True} \mid \text{Event}\textsubscript{B} == \text{True}) == 66%

Temporal

Event\textsubscript{A}, Event\textsubscript{B}, Event\textsubscript{A}, Event\textsubscript{B}, … == True
Invariants - suggested complement or alternative to 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 ];
    }

    ...
}
**Approach -- alternative slide to show algorithm**

Expand upon existing inference techniques

```python
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

<table>
<thead>
<tr>
<th>Trace</th>
<th>Pattern</th>
<th>Confidence</th>
<th>Instantiated</th>
<th>Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

matrixD.det = 0
matrixD.det = -6
matrixD.det = 2
matrixD.det = 2
matrixD.det = 2
Problem Space

- Scalars
- Vectors
- Others
- Var space
- Rel Var
- Language
- Environments