Next-Generation Software Verification and Adaptation: An AI Perspective

Shiva Nejati

snejati@uottawa.ca, @ShivaNejati

Sedna Lab, School of Electrical Engineering and Computer Science (EECS)

uOttawa

Software testing is about finding failures, assuming that failures are due to faults in the system under test (SUT)

What if failures happen but the SUT is not faulty?

A failure may indicate insufficiencies such as performance limitations, physical constraints, or misuse by human operators

This talk proposes to use Interpretable Machine Learning to learn insufficiencies caused by the SUT environment

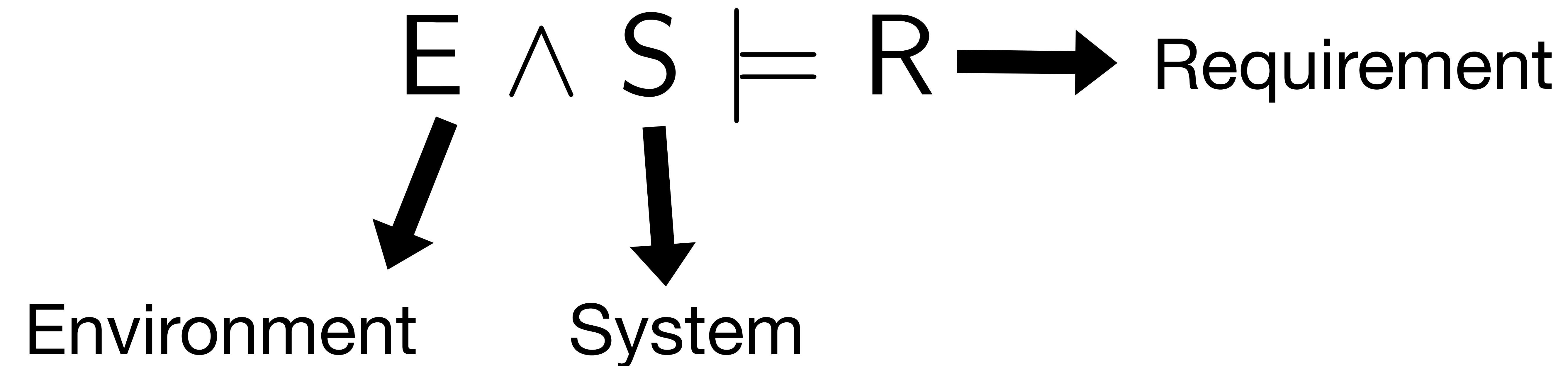
The meaning of requirements
(Michael Jackson)

- Requirements are located in the *environment*, which is distinguished from the *machine* to be built. A requirement is a condition over *phenomena* of the environment. A *specification* is a restricted form of requirement, providing enough information for the implementer to build the machine (by programming it) without further environment knowledge.

Michael Jackson: *The Meaning of Requirements*. Annals of Software Engineering 3: 5-21 (1997)

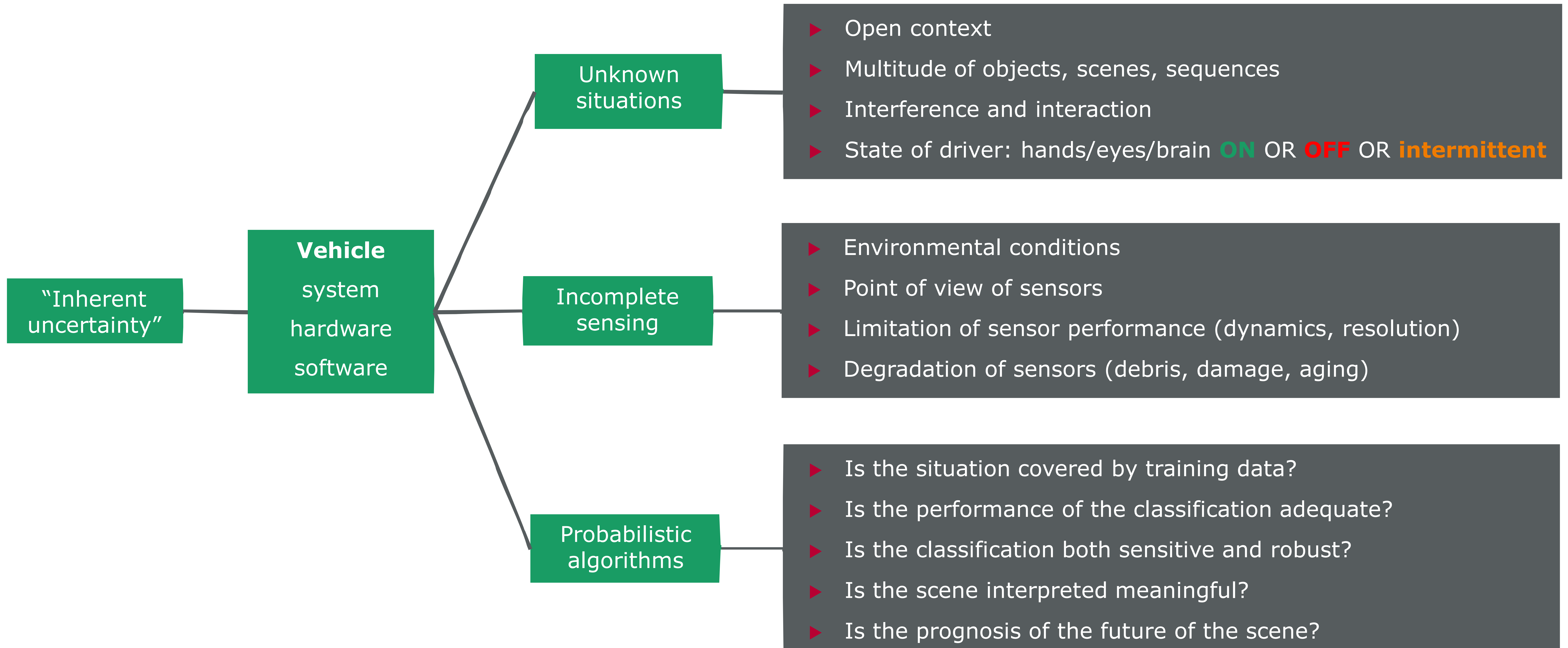
Verification

Demonstrating correctness of all system environments (usages):



Verification fails when there is a fault in **S,** or there is a mismatch between **S** and **E**

Environment and System Mismatches



Failure taxonomy for Safety of the Indented Functionality (SOTIF) for self-driving systems

©2019 Vector Consulting Services GmbH

Testing

Checking the system for some normal and boundary usages (environments)

$$\exists E \cdot E \land S \land \neg R$$

Understanding the environment conditions leading to failures is becoming essential

- E.g., the chance of an accident is higher for an emergency breaking system when the car drives with a speed higher than 30km/h on a curved road (> 60°)

Repair or Adaptation

- We observe a failure for some environment (**E***)

$$E^* \land S \land \neg R$$

- We modify the system to resolve the failure

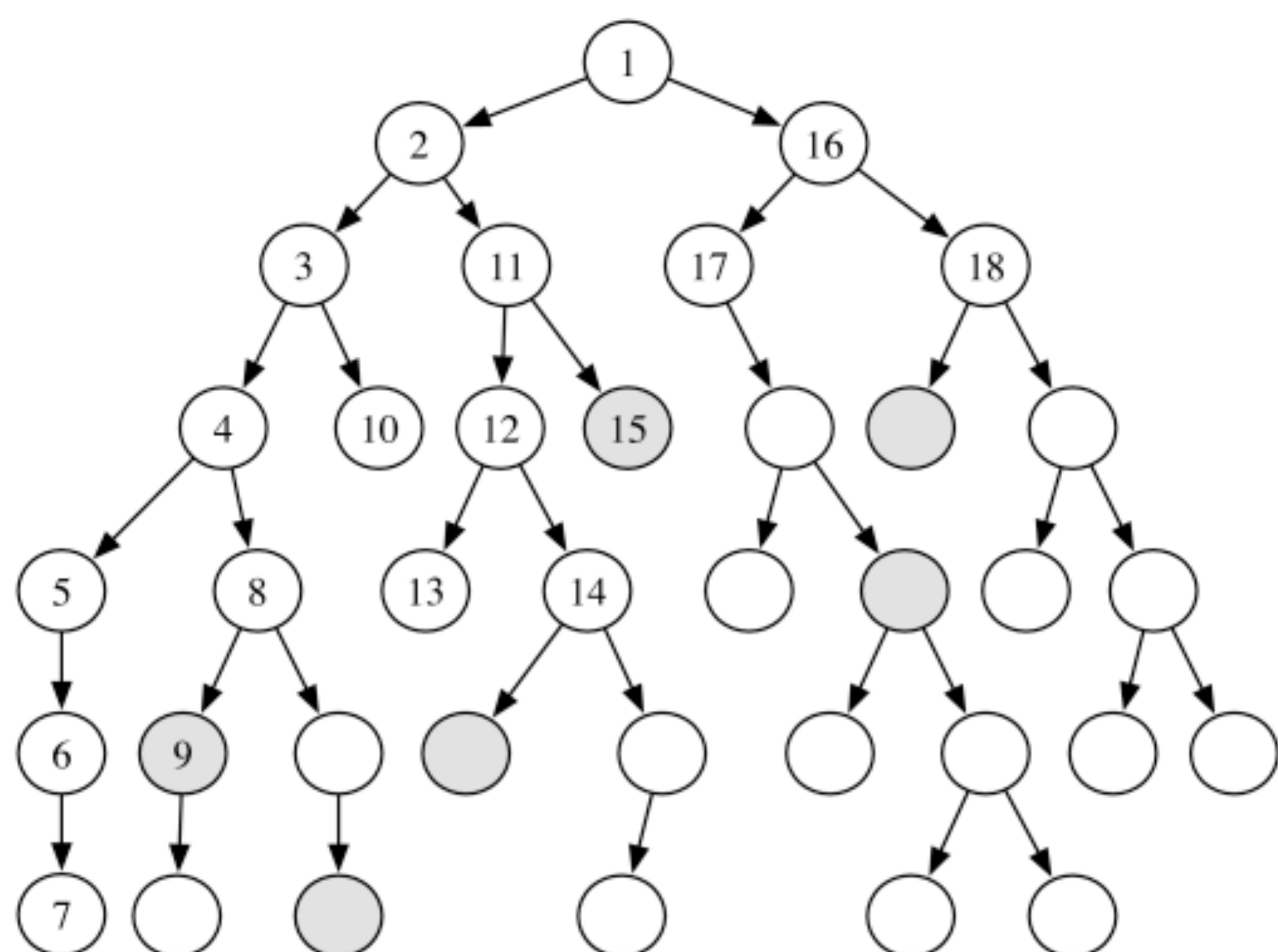
$$E^* \land S^* \rightarrow R$$

- But the focus of repair might be limited to a particular environment (**E***)
  - Overfitting

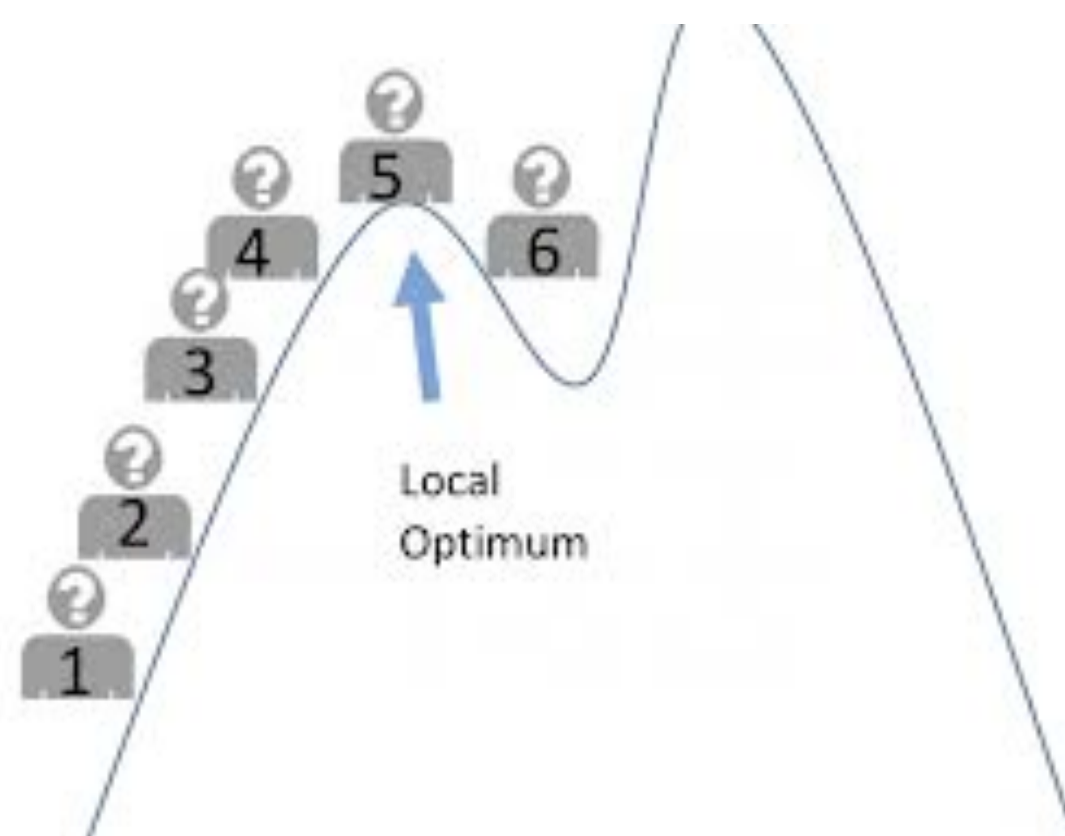
A Search Problem

- The core of formal verification, testing and adaptation is a search problem

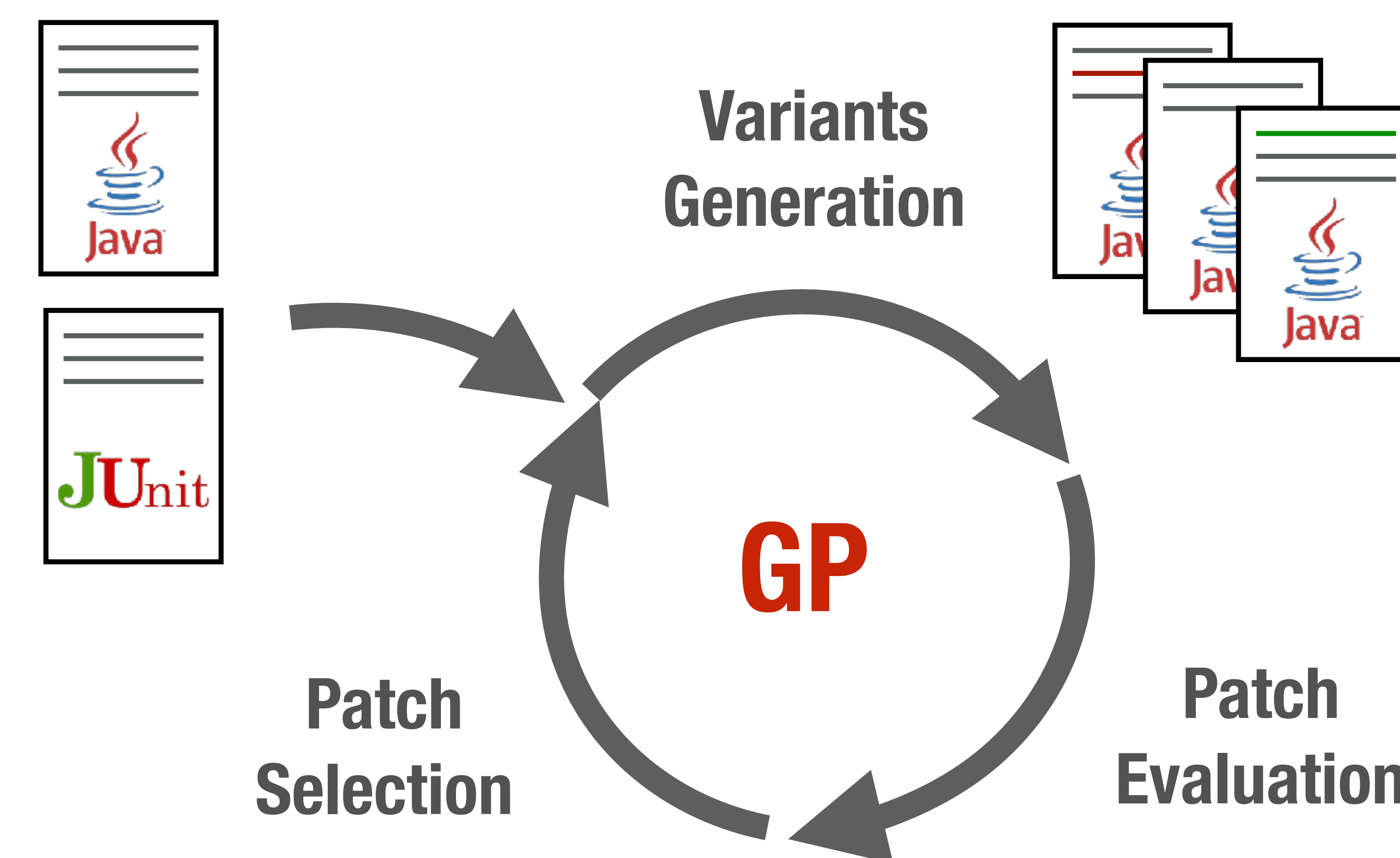
Verification
(Exhaustive Search)



Testing
(Heuristic Search)



Adaptation/Repair
(Heuristic Search)



Broadening the Search to Include Learning

- While we search, we should learn:
  - environment constraints that ensure satisfaction of requirements

$$\forall E \cdot E \wedge S \models R$$

  - environment conditions explaining failures

$$\exists E \cdot E \wedge S \wedge \neg R$$

  - repair/adaptation strategies valid for varying environments

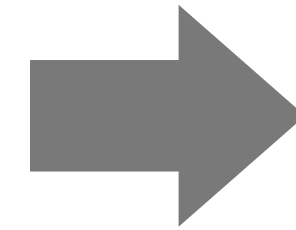
$$E^* \wedge S^* \rightarrow R$$

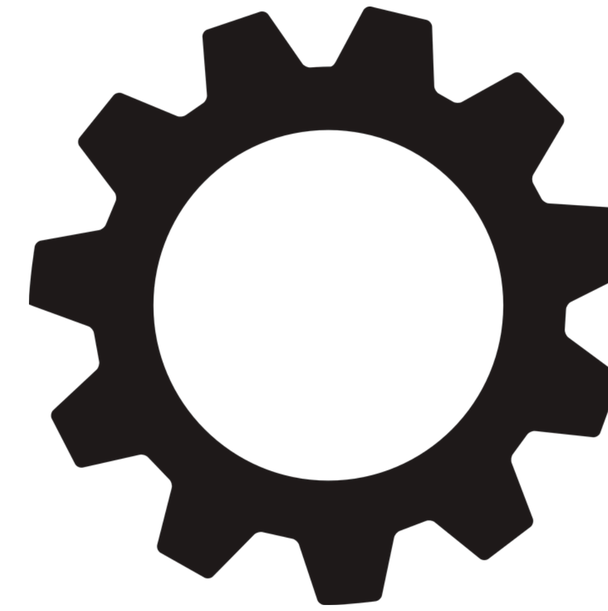
Broadening the focus of verification, testing and adaptation to learn about the SUT environment.

to learn about the SUT environment.

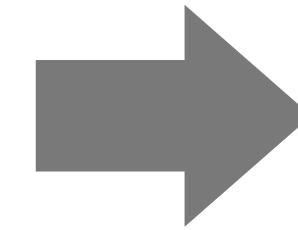
$\forall E \cdot E \wedge S \models R$

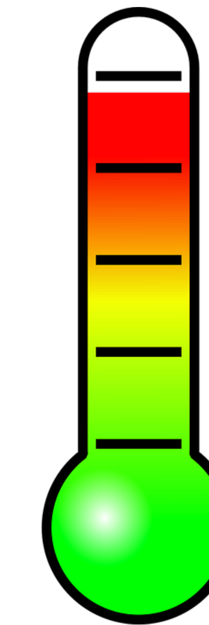
1	3	0.8	100
0	5	0.4	150



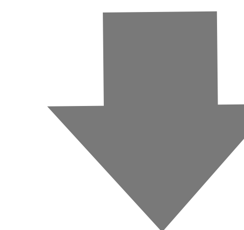


Executable System/Simulator





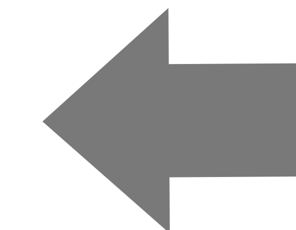
Satisfaction Measure, or Test Oracle



1	3	0.8	100	0.6
0	5	0.4	150	-2.6

or

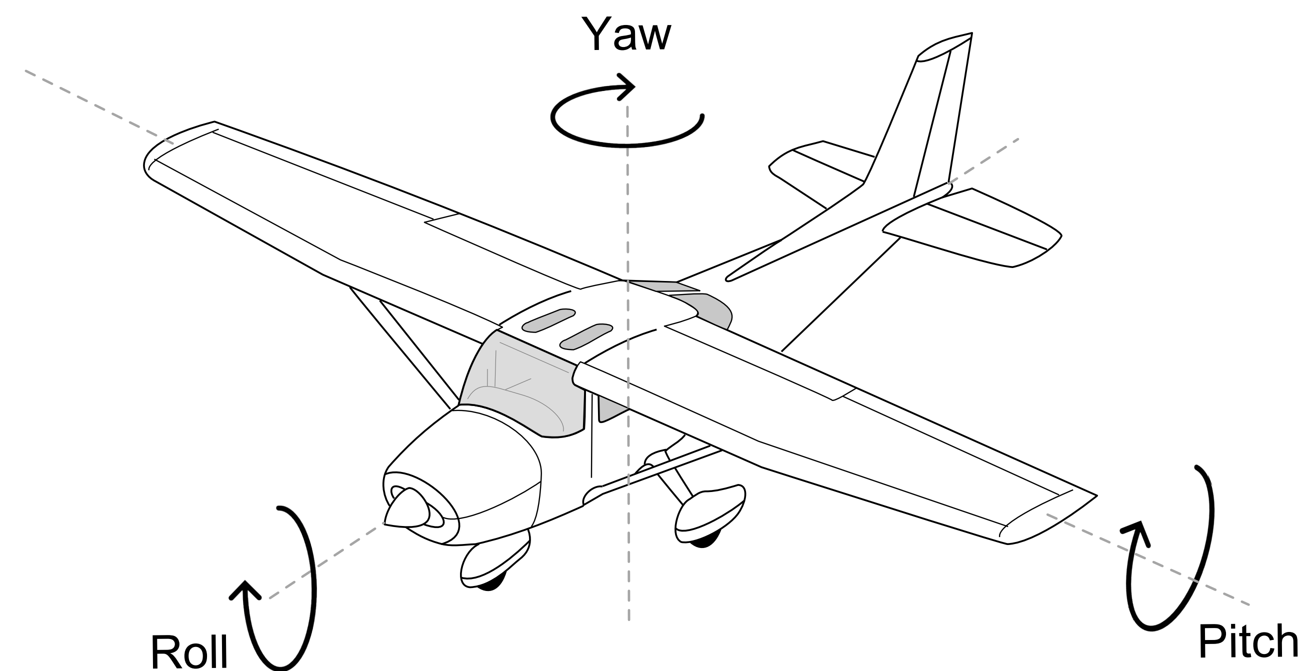
0	5	0.4	150	F
1	3	0.8	100	P



We use Interpretable ML to learn

- Constraints (assumptions) on environment to ensure satisfaction of some requirements
- Environment conditions leading to failures
- Fixes/adaptations that are valid for several environments

Mining Assumptions using Interpretable ML

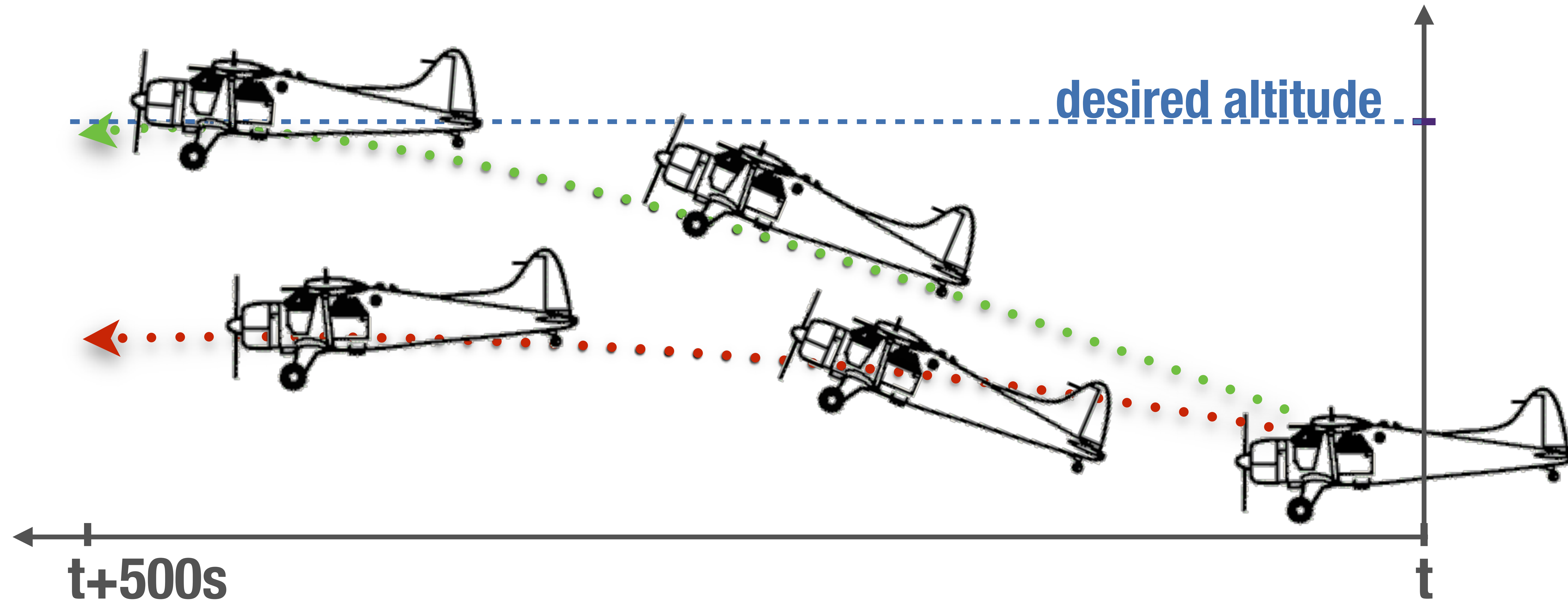


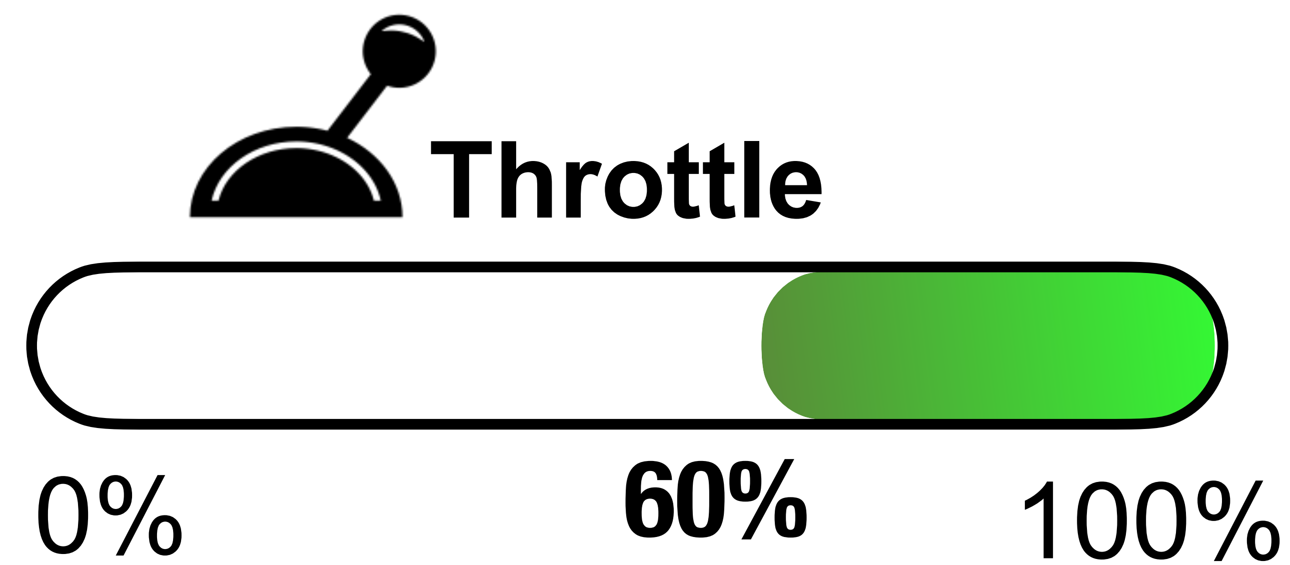
FAIL

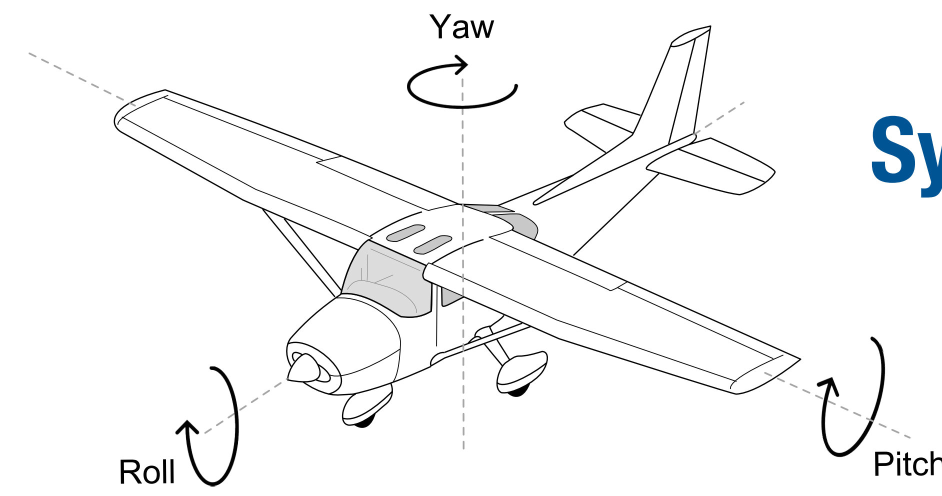
Req: When the autopilot is enabled, the aircraft should reach the desired altitude within 500 seconds in calm air.

Missing assumption!

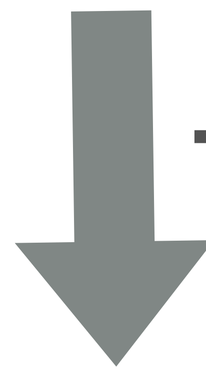
Autopilot Case Study



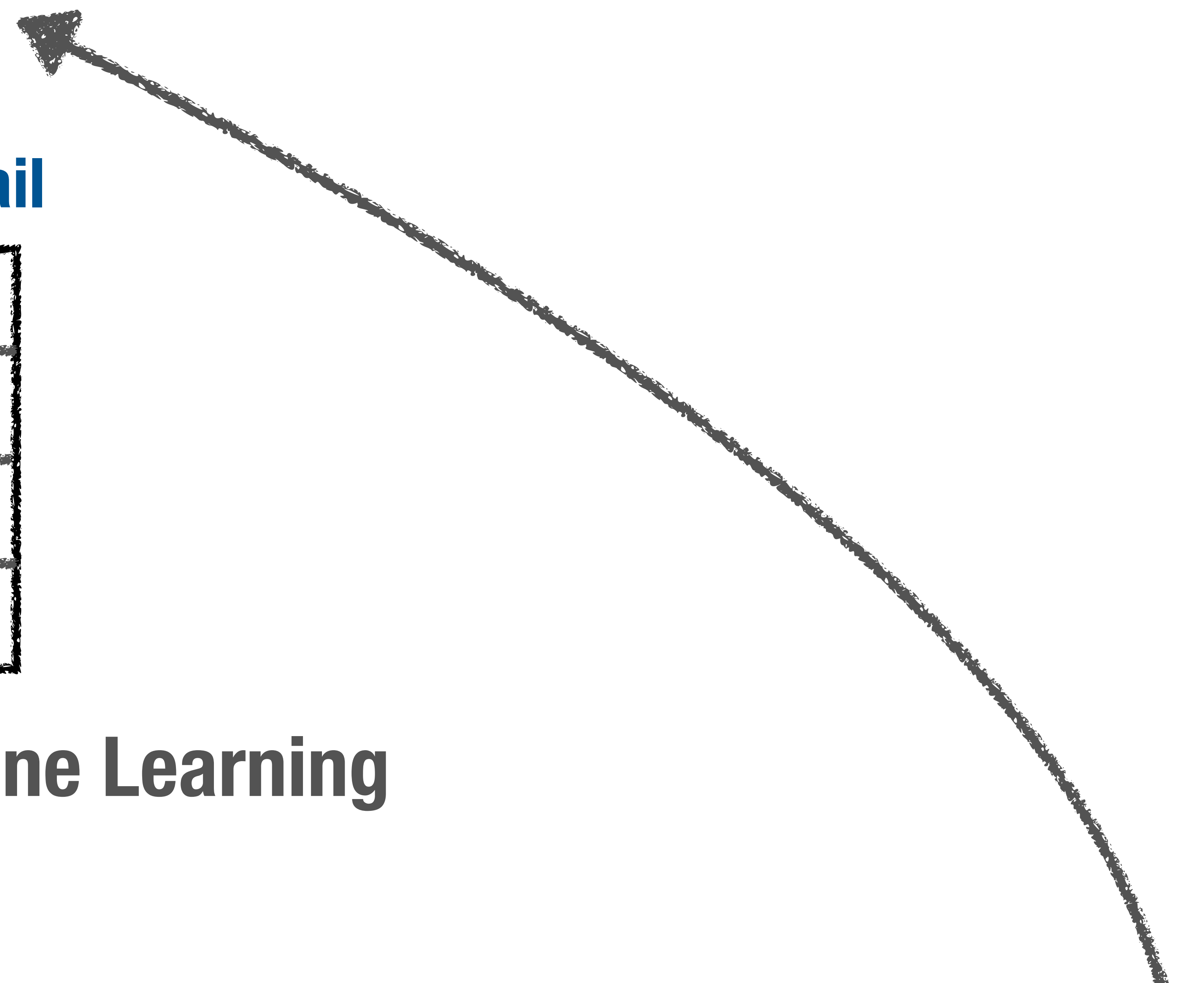




System + Requirement

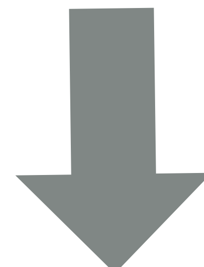


Test Generation

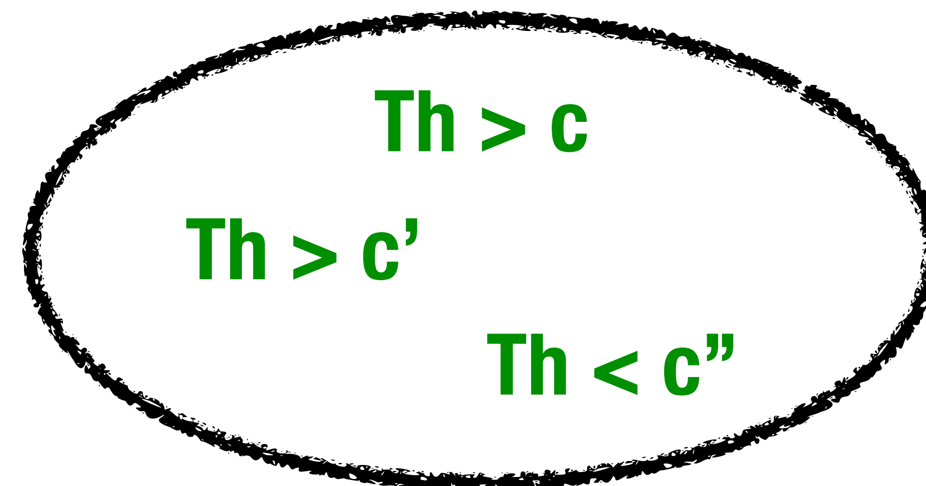


Test Inputs + pass/fail

Th = 20	P
Th = 0.4	F
Th = -3.6	F
Th = 100	P



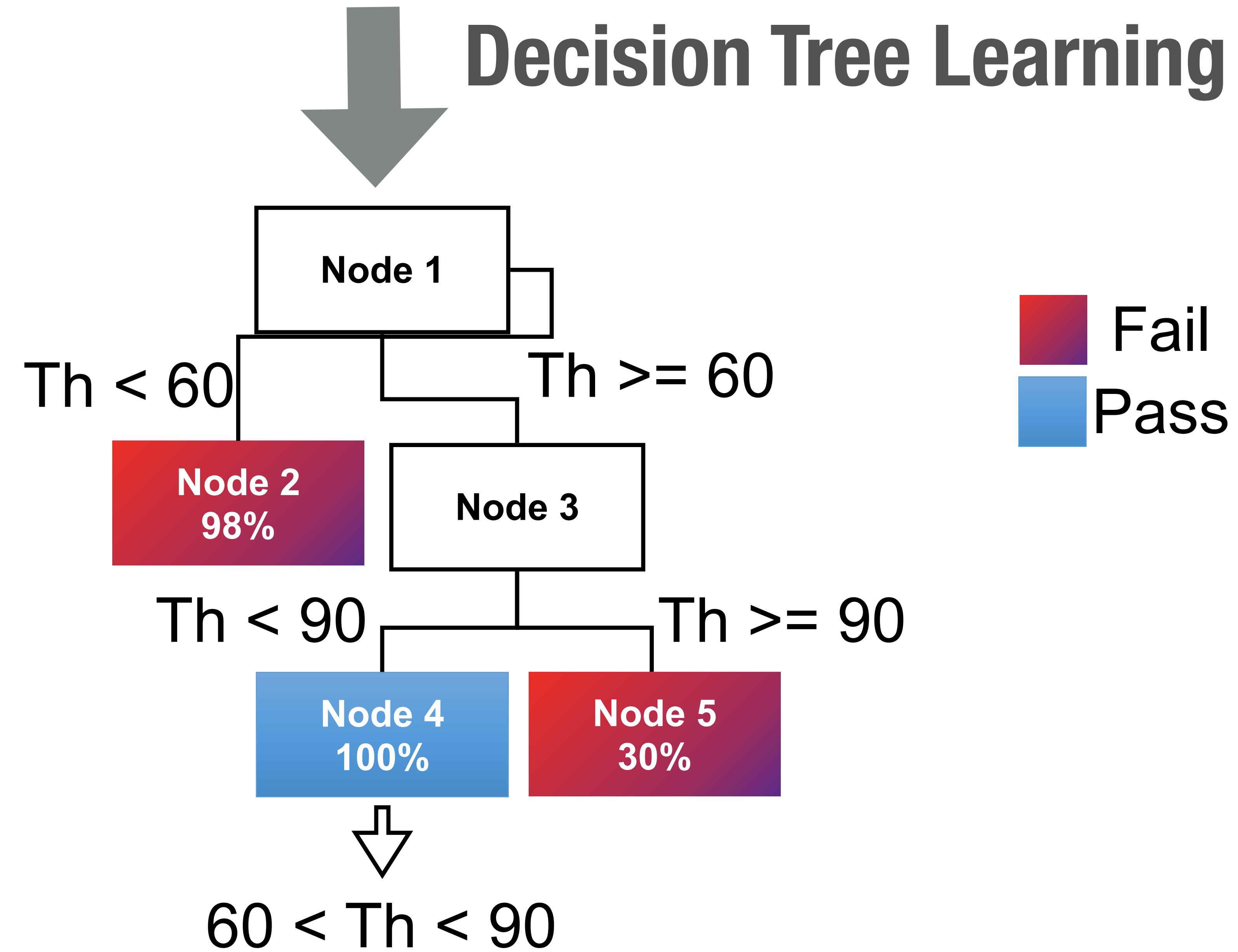
Machine Learning



Candidate Environment Assumptions

Model Checking

OR



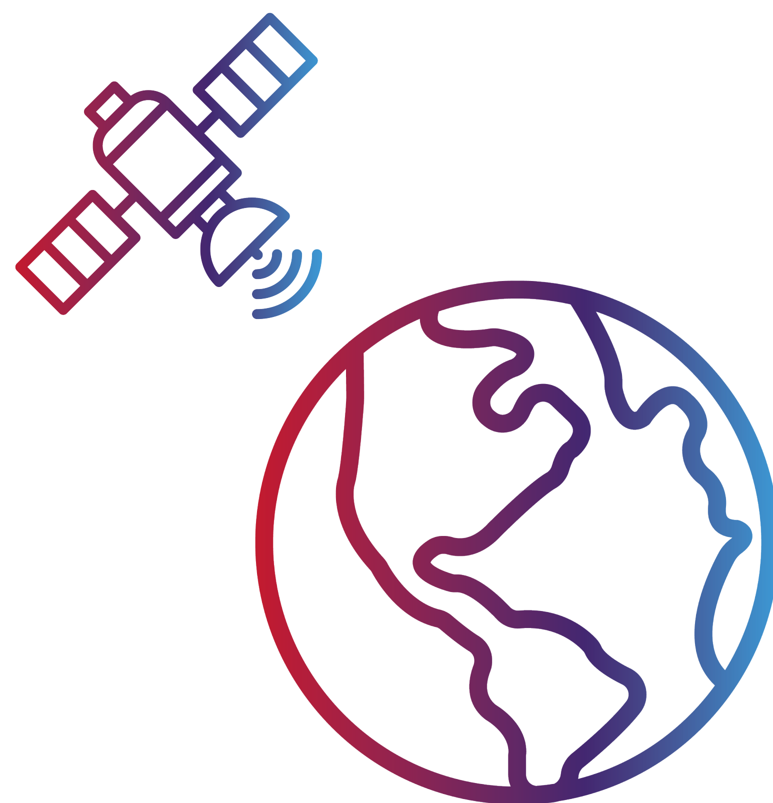
Assumptions learned by decision trees

- **Simple** conditions
- **Low** coverage

Khouloud Gaaloul, Claudio Menghi, Shiva Nejati, Lionel C. Briand, David Wolfe: *Mining assumptions for software components using machine learning.* ESEC/SIGSOFT FSE 2020: 159-171

ESAIL Case Study - Attitude Control Component

"When the **norm** of the **attitude error quaternion** is lower than **0.001**, the **torque** commanded to the reaction wheel around the x-axis shall be within **[-0.001, 0.001]N.m**"



$$\mathtt{A}_1 ::= \forall t \in [0,1] : (\overbrace{\exp(t) + 1 \geq 0}^{\mathtt{P}_1(t)} \wedge \overbrace{\exp(t) - 1 \leq 0}^{\mathtt{P}_2(t)})$$

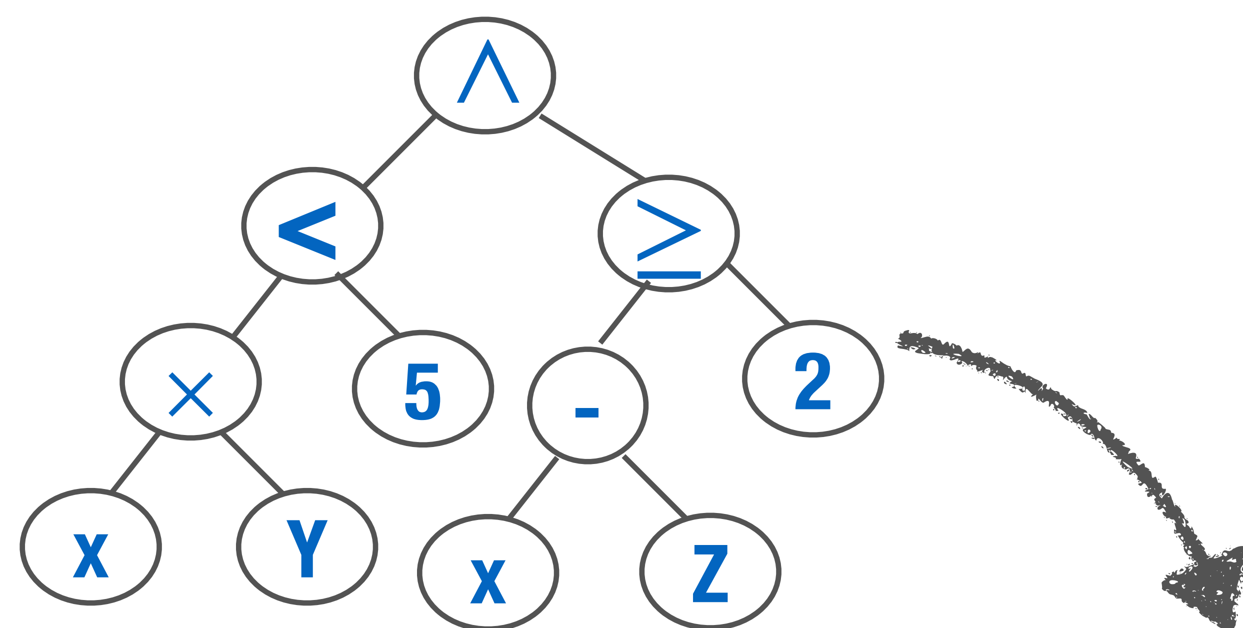
where:

$$\begin{aligned}\exp(t) = \quad & \overbrace{+783.3 \cdot \omega_e_\mathbf{x}(t)}^{\mathtt{T1}} \overbrace{-332.6 \cdot \omega_e_\mathbf{y}(t)}^{\mathtt{T2}} \overbrace{+3.5 \cdot \omega_e_\mathbf{z}(t)}^{\mathtt{T3}} \\ & - 50.57 \cdot \omega_e_\mathbf{x}(t) \cdot \omega_e_y(t) - 4751.8 \cdot \omega_e_x(t) \cdot \omega_e_z(t) \\ & + 3588.7 \cdot \omega_e_y(t) \cdot \omega_e_z(t) \\ & + 1000 \cdot \mathtt{Rwh_y}(t) \cdot \omega_e_z(t) - 1000 \cdot \mathtt{Rwh_z}(t) \cdot \omega_e_\mathtt{y}(t) \\ & \underbrace{-54.8 \cdot \omega_e_y(t)^2}_{\mathtt{T9}} \underbrace{+54.8 \cdot \omega_e_z(t)^2}_{\mathtt{T10}}\end{aligned}$$

Test Inputs + pass/fail

x = 5.6	Y = 20	z = 120	P
x = 0.4	Y = -20	Z = 110	F
x = -3.6	Y = 0	Z = 111	F
x = 2.6	Y = 10	Z = 109	P

Genetic Programming



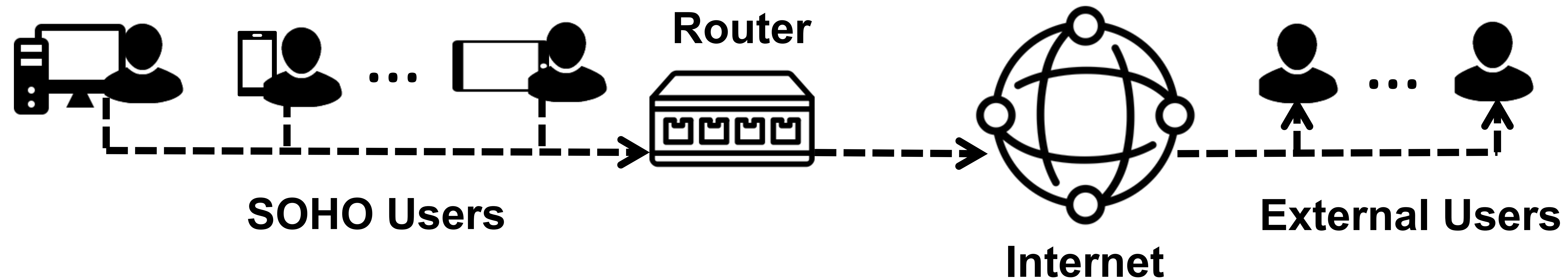
$$x \times y < 5 \land (x - z) \geq 2$$

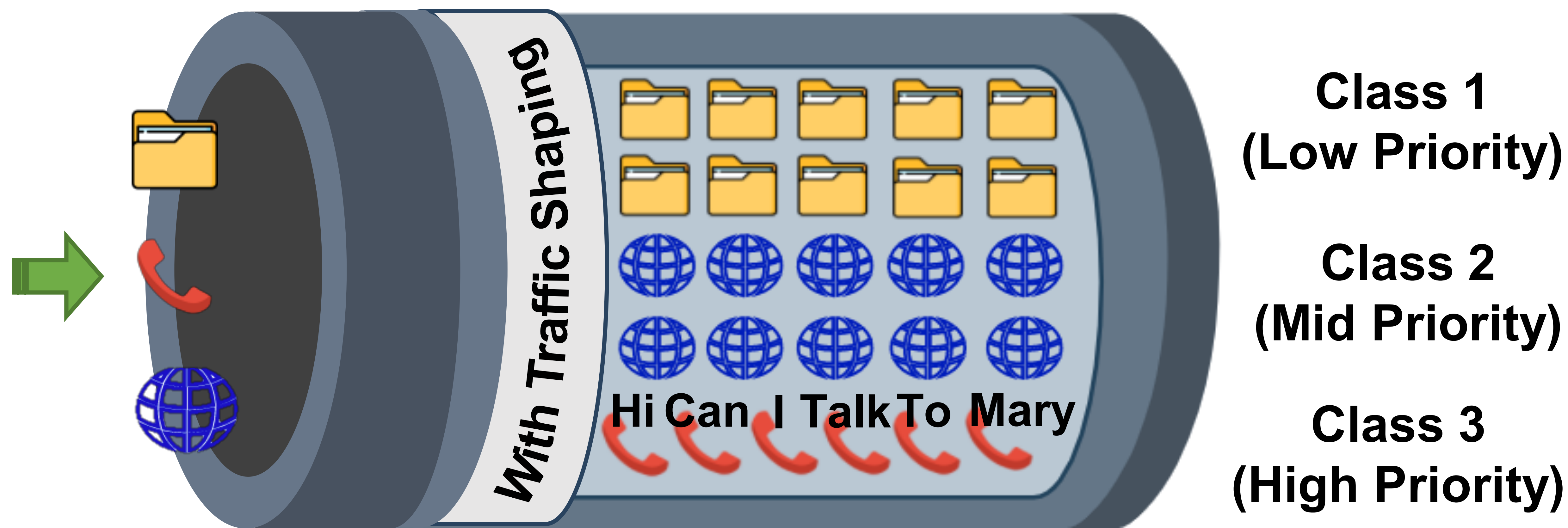
Complex linear and nonlinear formulas

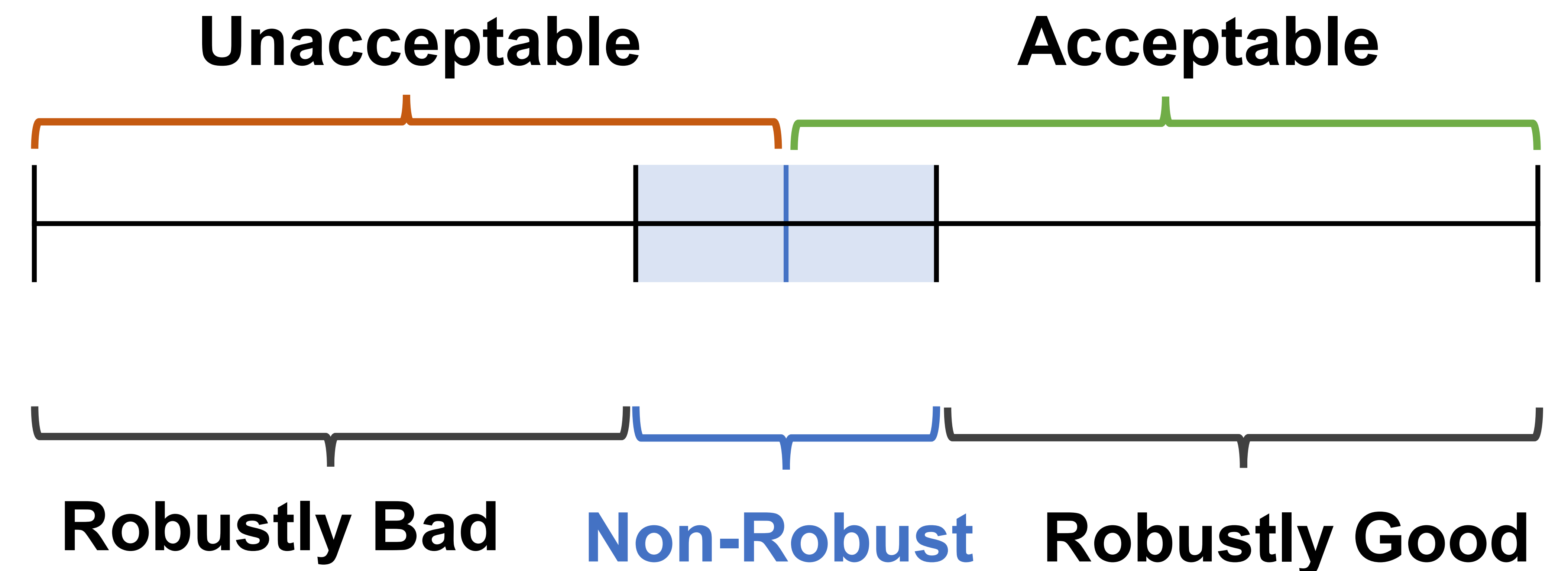
Khouloud Gaaloul, Claudio Menghi, Shiva Nejati, Lionel C. Briand, Yago Isasi Parache: *Combining Genetic Programming and Model Checking to Generate Environment Assumptions*. IEEE TSE, 2022

Learning Circumstances of Failures

A traffic management algorithm that provides Small Office/Home Office (SOHO) users with high quality connectivity for streaming applications (e.g., Zoom).



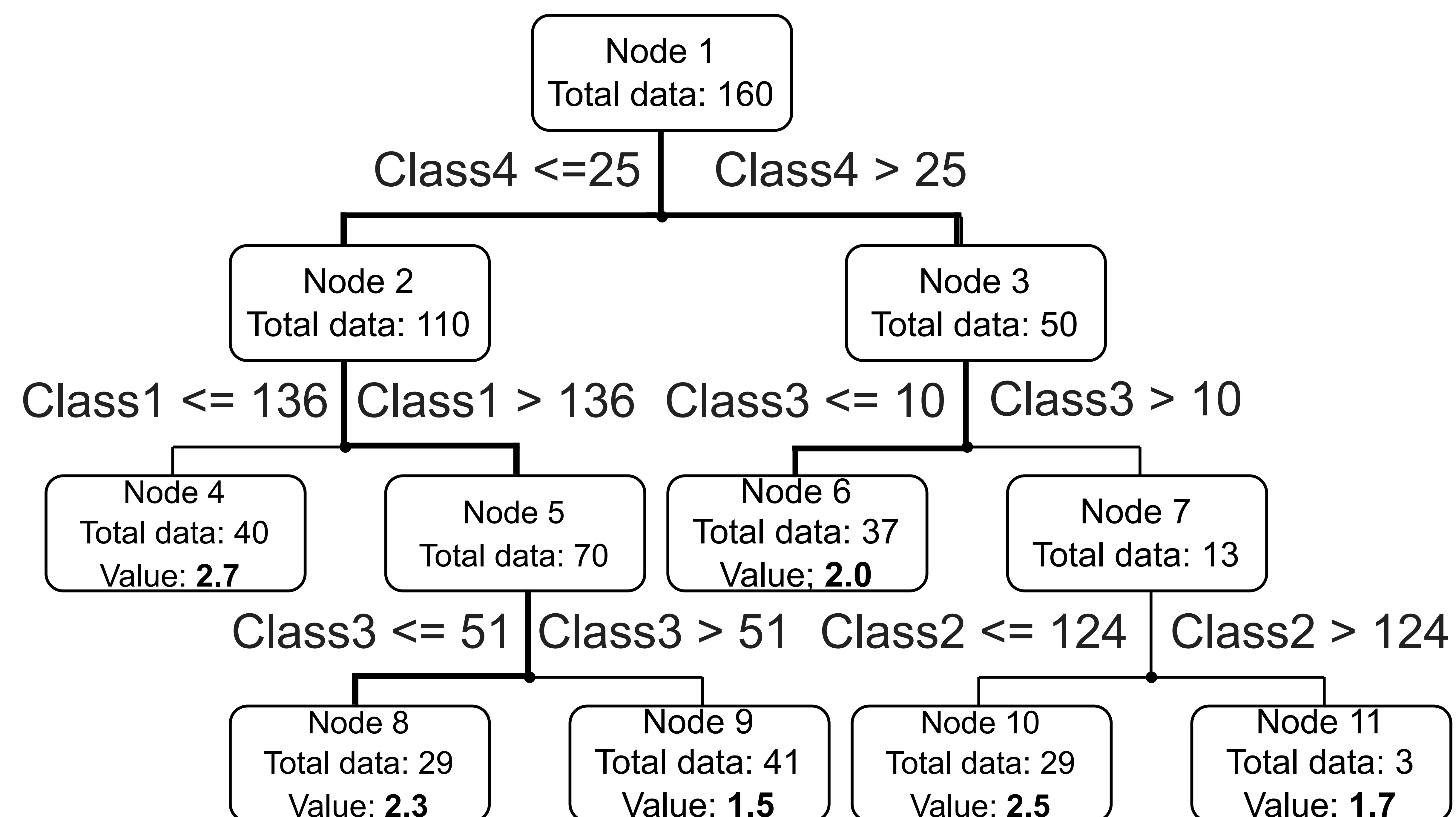




Goal: Understanding conditions over the bandwidths of different priority classes leading to non-robust connectivity.

Test Generation and Learning

1. Build a regression tree
2. Find paths that are closest to the boundary
3. Identify variables on the paths and constraints for the variable ranges
4. Generate more test cases in the identified ranges, rebuild the tree, and go to step 2.

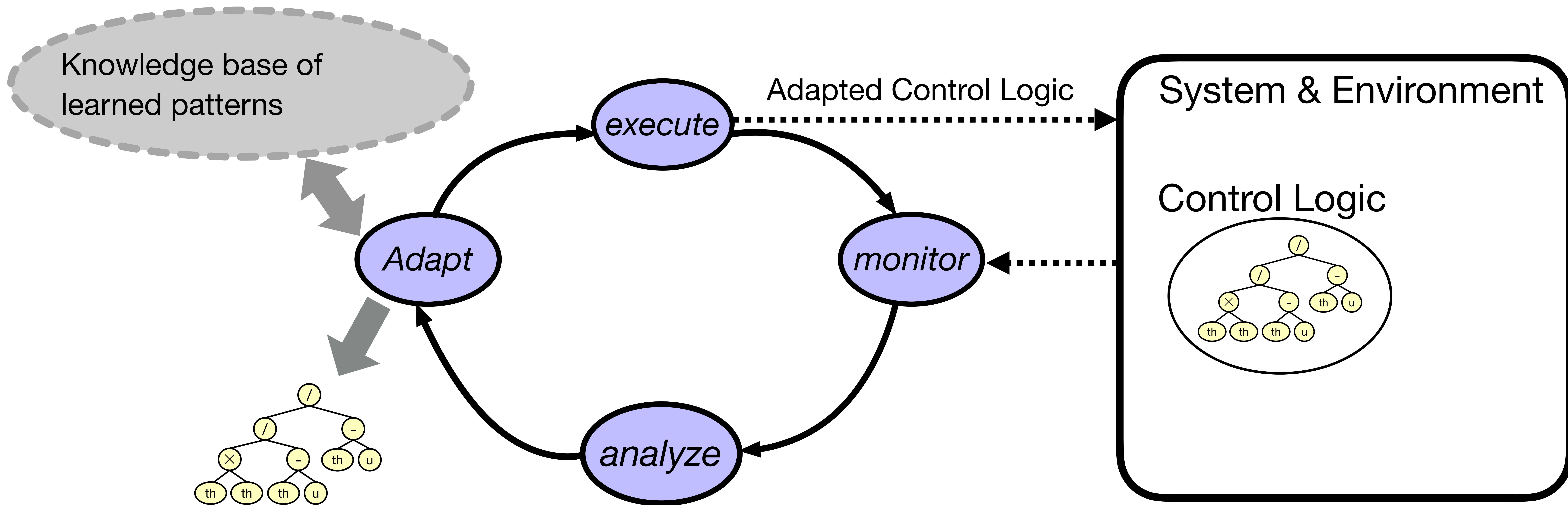


Input ranges leading to non-robustness:

Class1: [18%TB , 28%TB]
Class2: [32%TB , 42%TB]
Class3: [6%TB , 16%TB]
Class4: [0%TB , 7%TB]
**TB: Total Bandwidth

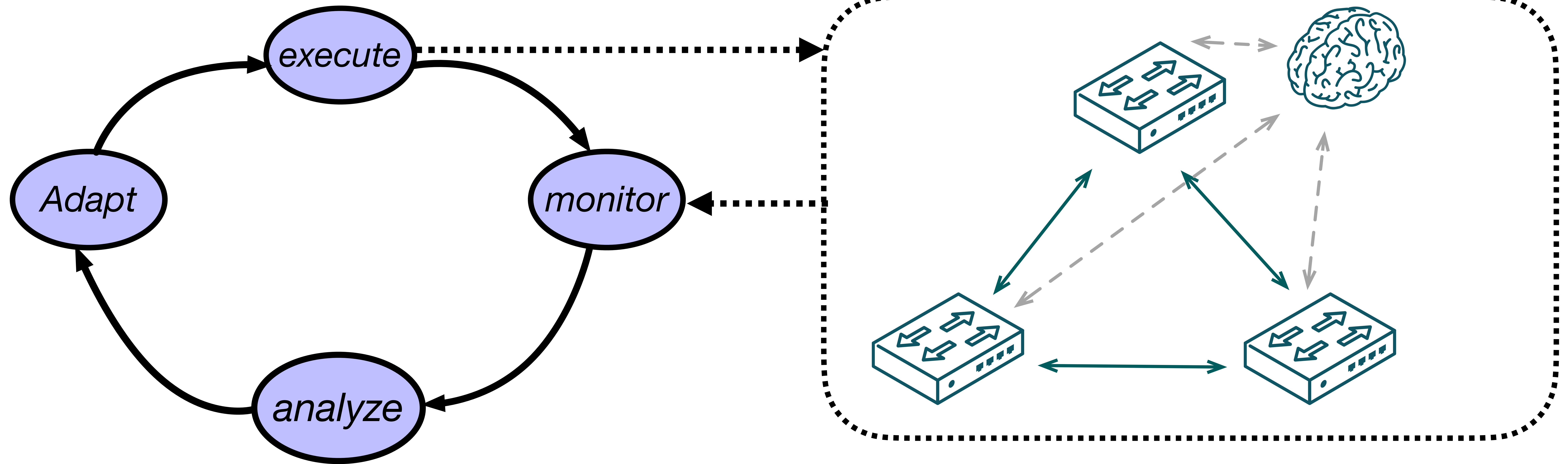
Baharin A. Jodat, Shiva Nejati, Mehrdad Sabetzadeh, Pat Saavedra, Learning Non-robustness using Simulation-based Testing: a Network Traffic-shaping Case Study. Under submission.

Self-Adaptation through Genetic Improvement of Software



Improve using Genetic Programming
until the observed anomaly is resolved

Self-Adaption for Network Controllers



Jia Li, Shiva Nejati, Mehrdad Sabetzadeh: Learning Self-adaptations for IoT Networks: A Genetic Programming Approach. SEAMS 2022: 13-24

Conclusions

- For system-level testing, it is not difficult to find test scenarios leading to failures, and failures may not be due to a (traditional) fault in the SUT
- Understanding the environment conditions leading to failures is becoming essential
- We propose to shift the focus of verification and testing from the SUT to the SUT’s environment
- As we perform search, we use interpretable ML to learn functional insufficiencies of the SUT