Fault Diagnosis for an Unknown Plant

Mohammad Mahdi Karimi
Second year PhD. Candidate, ECE
Supervisor: Dr Ali Karimoddini
November 2015
mmkarimi@aggies.ncat.edu

North Carolina Agricultural and Technical State University

Our Websites:
techlav.ncat.edu
accesslab.net

Motivation

Antares Failure during Orb-3 Launch (Oct 28, 2014):

- On October 28, 2014, 6:22 p.m. (EST), *Orbital ATK* launched its *Orb-3 cargo* from NASA's Flight Facility in Virginia.

- Just over 15 seconds into flight, an explosion in the Main Engine System (MES) occurred, causing the vehicle to lose thrust and fall back toward the ground.

MOTIVATION

Orb–3 Accident Investigation Report : October 9, 2015

- In
- C
- The IRT devel

Technical F	
TF-1	Th ser of
TF-2	Giv tes bea
TF-3	Th ade
TF-4	A anc
TF-5	Alt the
TF-6	Ba

Technical Recommendations		Finding(s) Addressed by Recommendation
TR-1		-2, TF-4,
TR-2		TF-4 -7, PF-8)
	implemented specific to planned Antares operations.	
TR-3	For future Antares missions, additional MES sensors and sensor filtering should be provided by Orbital ATK.	TF-3
TR-4	For future engine ATPs, sensors more suitable to the test environment should be utilized, and sensors should be better placed to understand and characterize engine performance.	TF-3
TR-5	Greater insight and verification activities should be implemented to ensure that cleanliness is consistent with engine (and other critical component) requirements.	TF-5
TR-6	To further reduce the likelihood of moisture intrusion into the	TF-5

Question:
Are we able to place sensor for every possible fault?

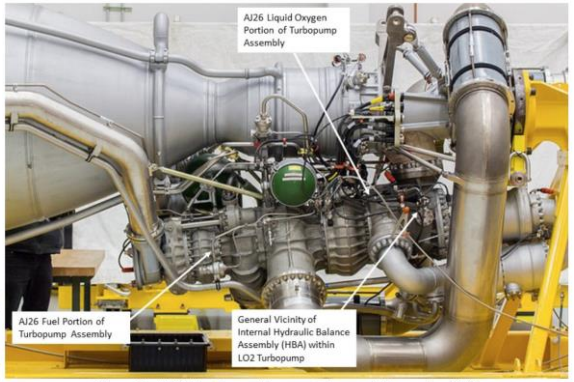


Figure 3. AJ26 Engine on Transportation and Processing Skid

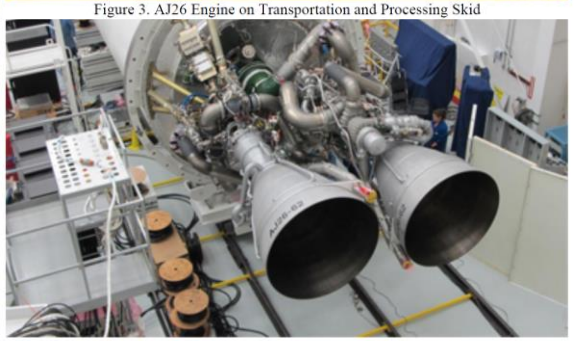


Figure 4. Antares First-Stage Core with Two AJ26 Engines Installed

National Aeronautics and Space Administration (NASA). "Nasa Independent Review Team Orb – 3 Accident Investigation Report (Executive Summary)." http://www.nasa.gov/sites/default/files/atoms/files/orb3_irt_execsumm_0.pdf.

MOTIVATION

Connection to TECHLAV Project

Thrust 1
(Modeling, Analysis and Control of Large-scale Autonomous systems of Vehicles (MACLAV))

Thrust 2
(Resilient Control and Communication of Large-scale Autonomous systems of Vehicles (RC2LAV))

Thrust 3
(Testing, Evaluation and Verification of Large-scale Autonomous systems of Vehicles (TEVLAV))

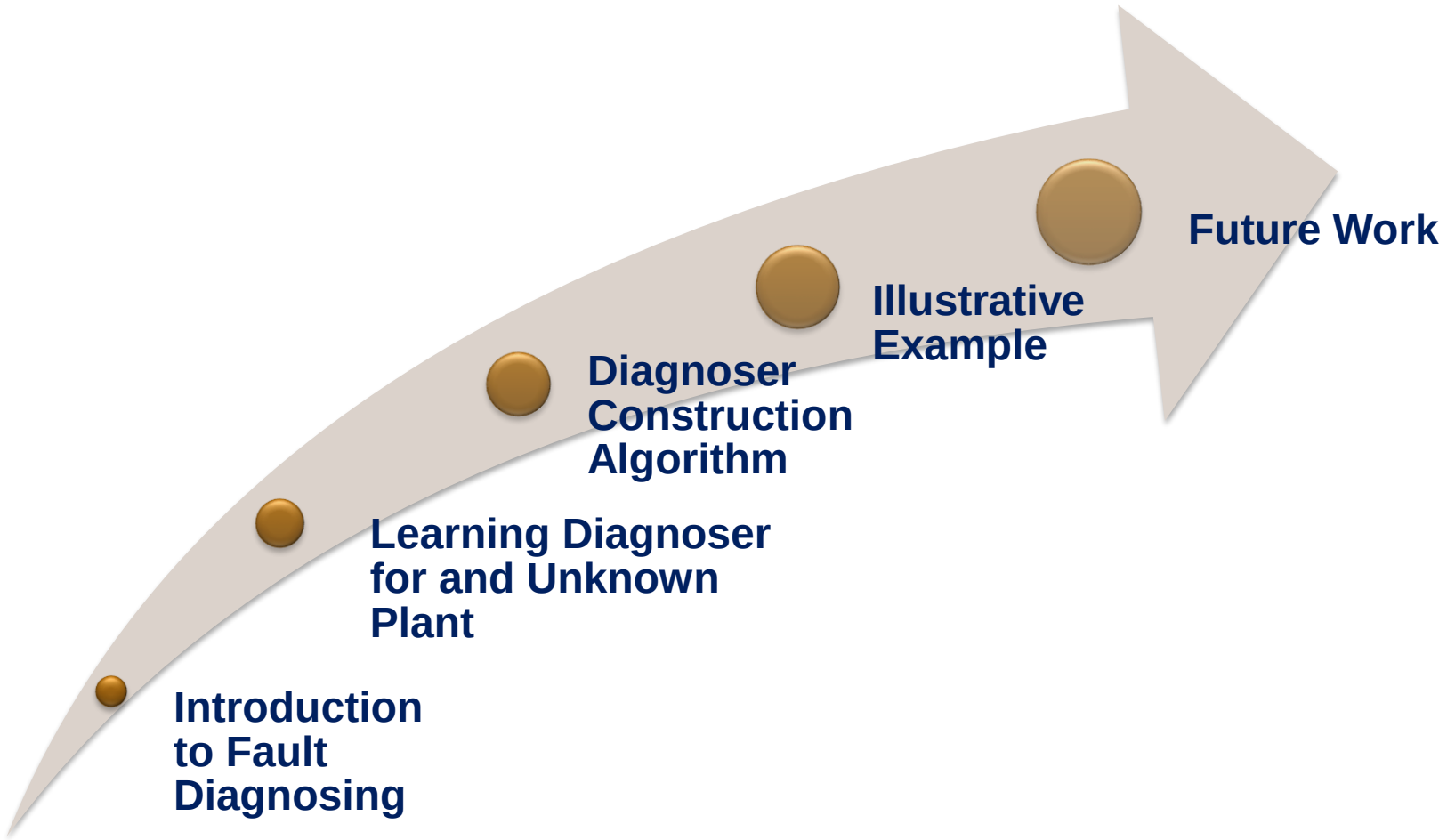
Developing systematic techniques, tools, and algorithms to enhance the reliability and efficacy of the control structure and the communication backbone for LSASVs integrated with human operators in dynamic and uncertain environments such as a battlefield.

Trust 2.1
Developing fault tolerant mechanisms for LSASV

Task T2.1:
Failure Diagnosis for Large-scale Autonomous systems of Vehicles (LSASVs)

Trust 2.2
Developing a reliable distributed communication network for LSASV

Outline


Introduction to Fault Diagnosing
Learning Diagnoser for and Unknown Plant
Diagnoser Construction Algorithm
Illustrative Example
Future Work

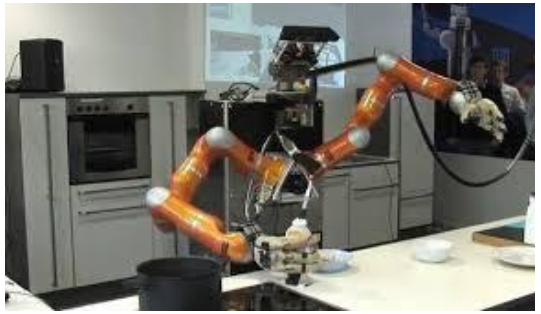
MOTIVATION

Facts:

- Despite all our efforts, failures in the system cannot be avoided.
- Failures may happen every where in a system.

There is a need to diagnose failures in the system: without using a sensor for every possible failure, try to detect failures from the behavior of the system.

Traditional fault diagnosis



Automatic fault diagnosis

Problem Description

Problem: In case of occurring a failure in the system, how to automatically diagnose a failure?

This problem can be broken down into three sub-problems:

Failure detection:	• Is a failure happened in the system?
Failure identification:	• What is the type of failure?
Failure location:	• Where is the place of failure in the system?

Objectives and Impacts

- Objectives

- To develop techniques for automatic diagnosis of failures in the system
- To timely diagnose (detect, identify and locate) occurred failures in the system.
- To diagnose failures in unknown dynamic systems

- Impacts

- By timely diagnosis of the failures in the system, there is a chance of recovering the system, and hence, improving the reliability of the system.

Challenges

- Faults may happen anywhere in the system causing despondent situations.
- Faults may happen anytime in the system causing despondent situations.
- Though we may use sensors for important possible failures, but practically we cannot have a sensor for every possible failure as failures may happen everywhere anytime.
- We might not always have access to the model of the system and its failures.
- Faults should be diagnosed in the shortest possible time to make it possible to be accommodated.

Proposed Approach

Assumption: Failures are abrupt changes in the system, and we may model failures in the system as "**events**".

Approach

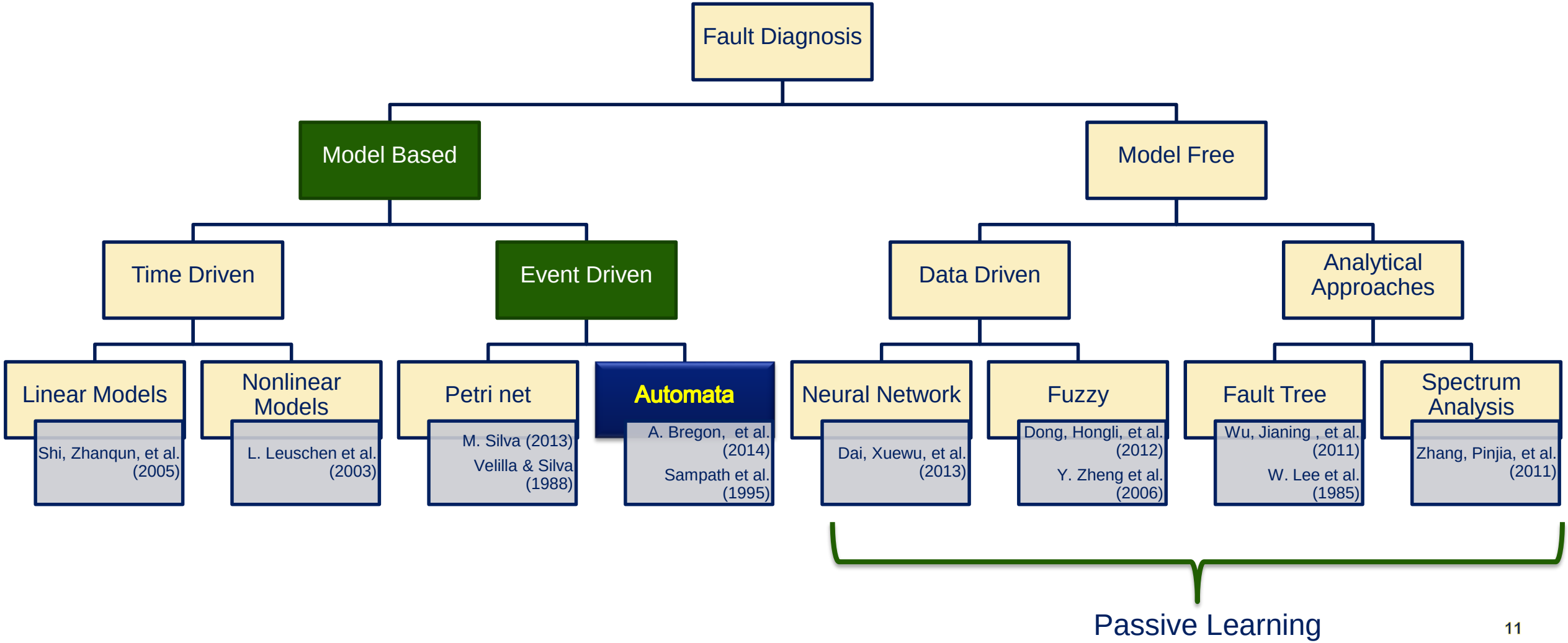
We use the theory of **Discrete Event Systems** to model the failures.

We develop a "**diagnoser**" as a diagnosis tool.

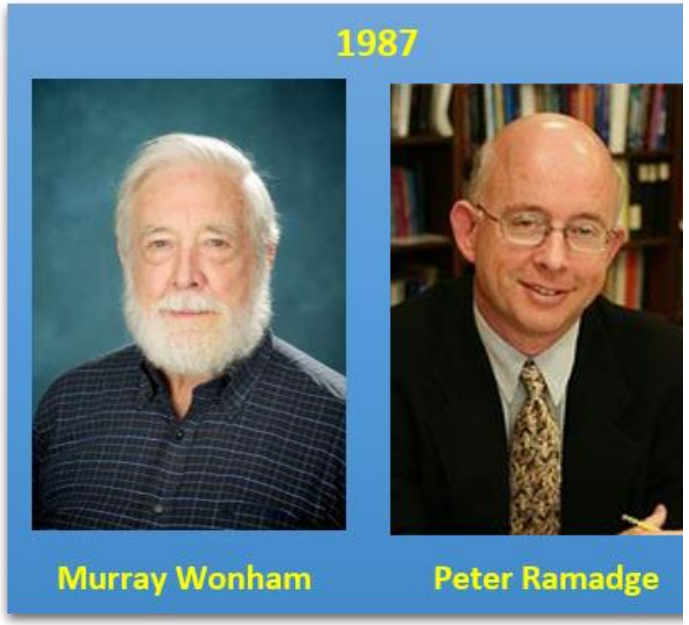
In the absence of complete information about the system, we develop an **active learning** technique to adaptively build-up a diagnosis tool for the system.

- M.M. Karimi, A. Karimoddini, A. P White, "*Event Driven Failure Diagnosis for an Unknown Plant*" American Control Conference (ACC), 2016, Boston, USA (*submitted*).

Background Review



Discrete Event Systems (DES)



The Control of Discrete Event Systems

PETER J. G. RAMADGE, MEMBER, IEEE, AND W. MURRAY WONHAM, FELLOW, IEEE

Invited Paper

A Discrete Event System (DES) is a dynamic system that evolves in accordance with the abrupt occurrence, at possibly unknown irregular intervals, of physical events. Such systems arise in a variety of contexts ranging from computer operating systems to the control of complex multimode processes. We survey a control theory for the logical aspects of such DESs. This theory was initiated by Ramadge and Wonham, and has subsequently been extended by the authors and other researchers to encompass control theoretic ideas such as controllability, observability, aggregation, and modular, decentralized, and hierarchical control. We concentrate on the qualitative aspects of control but also consider computation and the related issue of computational complexity.

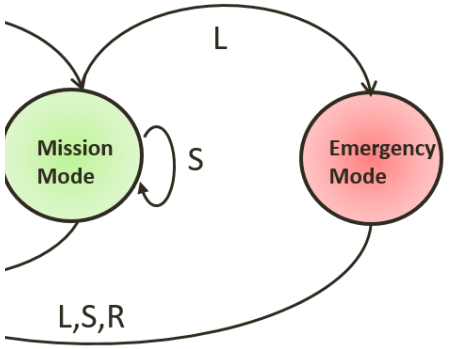
I. Introduction

A Discrete Event System (DES) is a dynamic system that evolves in accordance with the abrupt occurrence, at possibly unknown irregular intervals, of physical events. For example, an event may correspond to the arrival or departure of a customer in a queue, the completion of a task or the failure of a machine in a manufacturing system, transmission of a packet in a communication system, or the occurrence of a disturbance or change of setpoint in a complex control system. DESs arise in the domains of manufacturing, robotics, vehicular traffic, logistics (conveyance and storage of goods, organization and delivery of services), and computer and communication networks. These applications require control and coordination to ensure the orderly flow of events. As controlled (or potentially controllable) dynamic systems, DESs qualify as a proper subject for control theory, a viewpoint that we shall develop in some detail in the present paper.

In the past, DESs have usually been sufficiently simple that intuitive or ad hoc solutions to various problems have been adequate. The increasing complexity of man-made systems, made possible by the widespread application of complexity where more detailed formal methods become necessary for their analysis and design. Indeed, the use of VLSI has already led to the implementation of modular, hierarchical, and distributed systems on a scale never before possible. Such systems pose control and coordination problems of an unparalleled scope and complexity, and as yet there is little theory to serve as a guide in their resolution.

In this paper we survey automata and formal language models for DESs and the theory initiated by Ramadge and Wonham [56], [76], and subsequently extended by the authors and other researchers. The objective of this theory has been to examine control theoretic ideas such as controllability, observability, aggregation, and decentralized and hierarchical control for DESs from a qualitative viewpoint. The framework has proved useful in the theoretical analysis of a number of basic supervisory control problems [56], [76]; has motivated investigations using related models in database systems [26], [27] and manufacturing systems [38]; and, more recently, has been extended to cover modular [55], [95] and distributed [9], [35], [37] control. The main advantage of the model is that it separates the concept of open loop dynamics (plant) from the feedback control, and thus permits the formulation and solution of a variety of control synthesis problems. To date, the theory does not support all of the features that one would desire of a full theory of DESs. Nevertheless, the model has provided valuable concepts and insights to serve as guidelines for future work, and has contributed to our understanding of the fundamental issues involved in the analysis and control of DESs.

Computational problems for DESs are frequently complex. In our setting this manifests itself in the complexity of the computations involved in solving basic control syn-



Definition: A Discrete Event System (DES) is a dynamic system that evolves in accordance with the abrupt occurrence of physical events.

Why Discrete Event System?

To treat large-scale systems we need an abstract and simple tool

Many systems are inherently DES (Ex. Queuing system) and many others can be abstracted to a DES (Ex. Robotic, UAV)

DES can effectively model large-scale systems in an abstract mathematical fashion.

DES can capture system's logics and rules for constructing the decision making units and fault diagnoser for a system.

Discrete Event Systems (DES)

✓ In DES, **states** (operation modes) might change upon the occurrence of **events** (changes in sensor readings, commands).

DES

- Event – Based Models
  - In the event-based approaches, occurred events are being studied.
- State –Based Models
  - In state-based approaches, visited states are being studied.

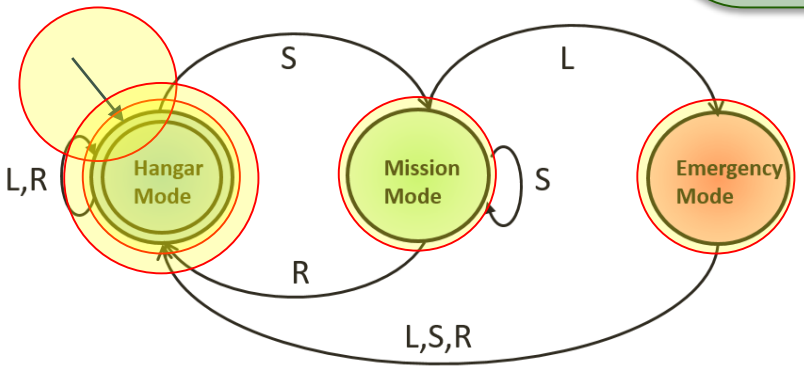
Automata

Automata is a graphical, mathematical useful tool to describe a DES.

Example:

$$G = (X, \Sigma, \delta, x_0, Xm)$$

- X: state space
- Σ: event set
- δ: partial transition
- x_0: Initial state
- Xm: final states



S: Start mission
R: Return to hangar
L: Low fuel alarm

$$\delta(HangarMode, S) = MissionMode$$

Problem Definition

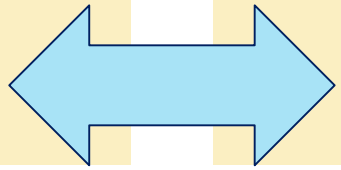
- Goals:

- To Detect, identify, and isolate faults timely in the system by monitoring performance or the external behavior of the DES system *G* modelled as an automaton :

$$G = (X, \Sigma, \delta, x_0)$$

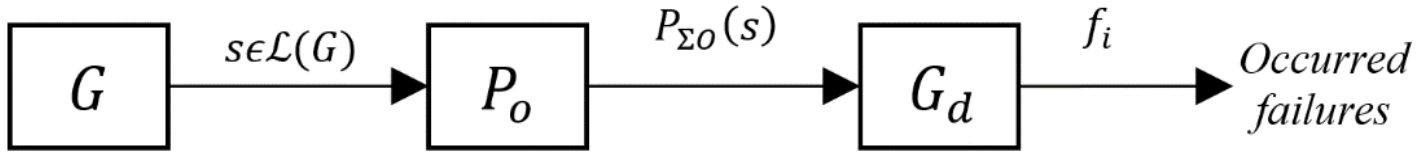
- Challenges:

- ❖ Faults maybe unobservable and we do not have dedicated sensor for every possible fault
- ❖ Usually systems are partially or completely unknown.



Our Solution

We develop a diagnoser to detect and isolate the failures.



Our developed Diagnoser **will determine if a failure has happened**

If yes, diagnoses the type of the occurred failure.

We do not know the system model. Instead, we develop an **active learning** method to build the diagnoser.

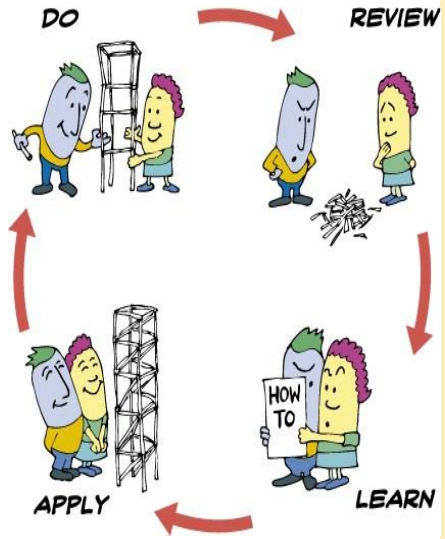
The proposing algorithm actively learns the diagnoser and the system within **DES framework** and builds the diagnoser $G_d = (Q_d, \Sigma_d, \Delta, \delta_d, h, q_0)$

Passive V.S. Active Learning

Passive Learning

- Teacher provides all available information about the system.
- Learner fits a model for the provided information.
- The learner passively learns the trained information and only can work over the training range.

Active Learning



- The learner asks basic questions about missing information about the system.
- The teacher answers the questions about the system.
- The learner actively learns the enquired information and gradually builds a model for the system.

Question:
How about the case that a new situation happens and the learner is not trained for it?

✓ The constructed model through an active learning procedure can gradually and adaptively construct a model for a system.

Contributions

We are using Discrete event system framework for fault diagnosing a completely unknown system.

A systematic active learning strategy is designed to construct diagnoser to provide diagnosis for an unknown system.

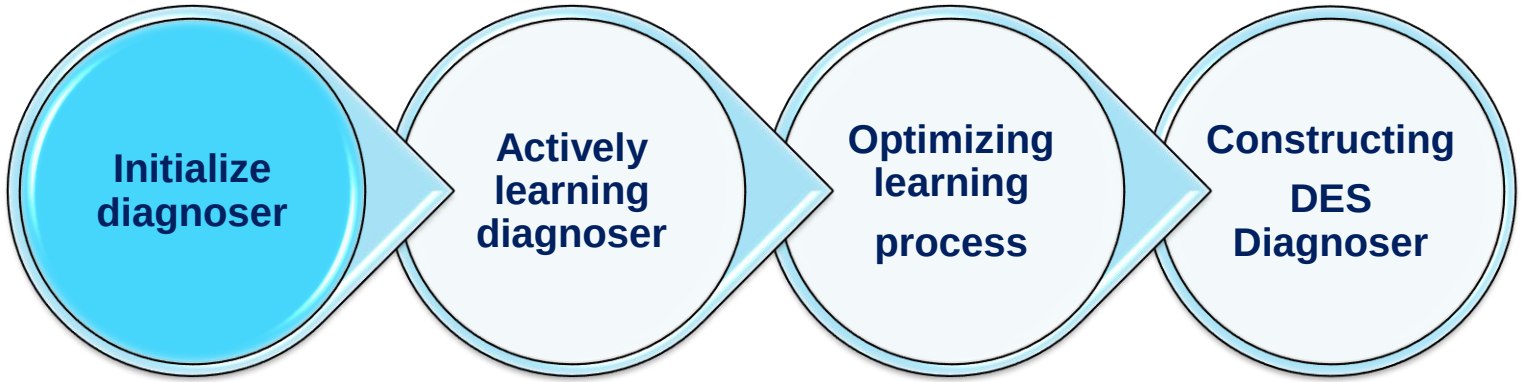
Actively asking basic minimum queries from an oracle, proposed algorithm will come up with a labeled deterministic finite state automaton as the system fault diagnoser.

An independent label propagation technique is designed to make the algorithm more efficient to construct the diagnoser.

Approach

✓ To address the fault diagnosis problem for unknown DES systems, we are proposing a new novel strategy to build a DES diagnoser through an active-learning process inspired by **L^*-learning**

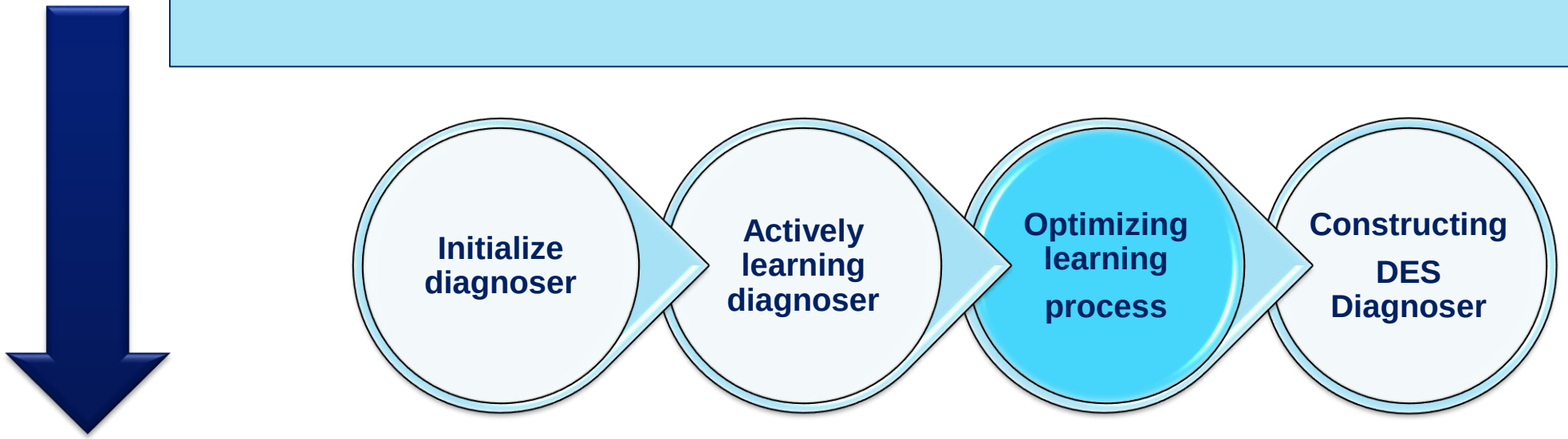
1. The proposed algorithm gradually learns the diagnoser starting with an initialized diagnoser, building up a deterministic label transition system for an unknown DES plant.



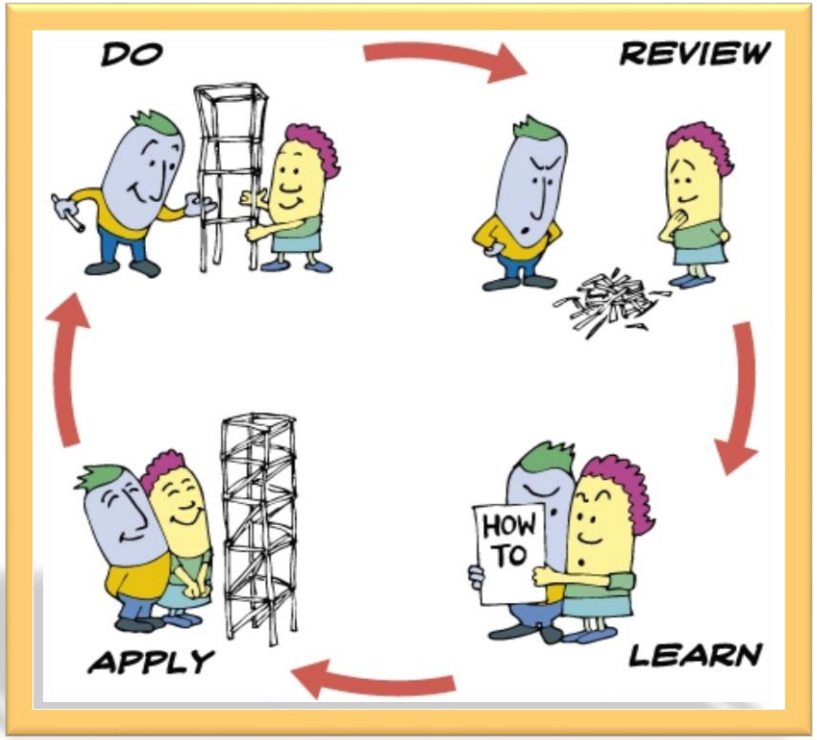
2. The proposed active-learning mechanism acquires the required information through an oracle who answers some basic queries about the system and observable strings.

Approach

3. A Label propagation method is introduced to make the fault diagnosing more efficient.



4. With the acquired information, the algorithm completes a series of observation tables, and eventually conjectures a correct diagnoser.



Section II:

THE PROPOSED ACTIVE-LEARNING ALGORITHM FOR CONSTRUCTING THE DIAGNOSER

Assumptions

Consider that in the plant *G*, failures $f_1, f_2, \ldots, f_n$ can happen:

1. Faults are unobservable:
   - We assume that these events are unobservable events in the automaton *G*, i.e.
     $$\Sigma_F = \{f_1, f_2, \ldots, f_n\} \subseteq \Sigma_{uo}$$
   - Otherwise, if failures are observable events, then they can be trivially and immediately diagnosed.
2. Failures make a distinct change in the system:
   - These changes (transition $x \xrightarrow{f_i} x'$) can be captured by the transition function $\delta(x, f_i) = x'$ in automaton *G*
3. Failures do not bring the system to a halt mode:
   - Therefore, there will be enough time to diagnose failures from the observable behavior of the system $P_{\Sigma_o}(\mathcal{L}(G))$.

Properties of DES Diagnoser

Diagnoser G_d can be described by a labeled automaton using the following tuple:

$$G_d = (Qd, \Sigma_d, \Delta, \delta_d, h, q_0)$$

Q_d is the set of diagnoser states	Δ is the output label set
$\Sigma_d = \Sigma_o$ is the event set	$h : Q_d \longrightarrow \Delta$ is the output function
δ_d is the transition rule	q_0 is the initial state

Output label set:

$$\Delta = \{N\} \cup \{L_1, L_2, \dots, L_m\}, L_i \in \{F_i, A_i\}$$

N: normal
F_i: occurrence of the failure f_i
A_i: ambiguity in the occurrence of the failure f_i

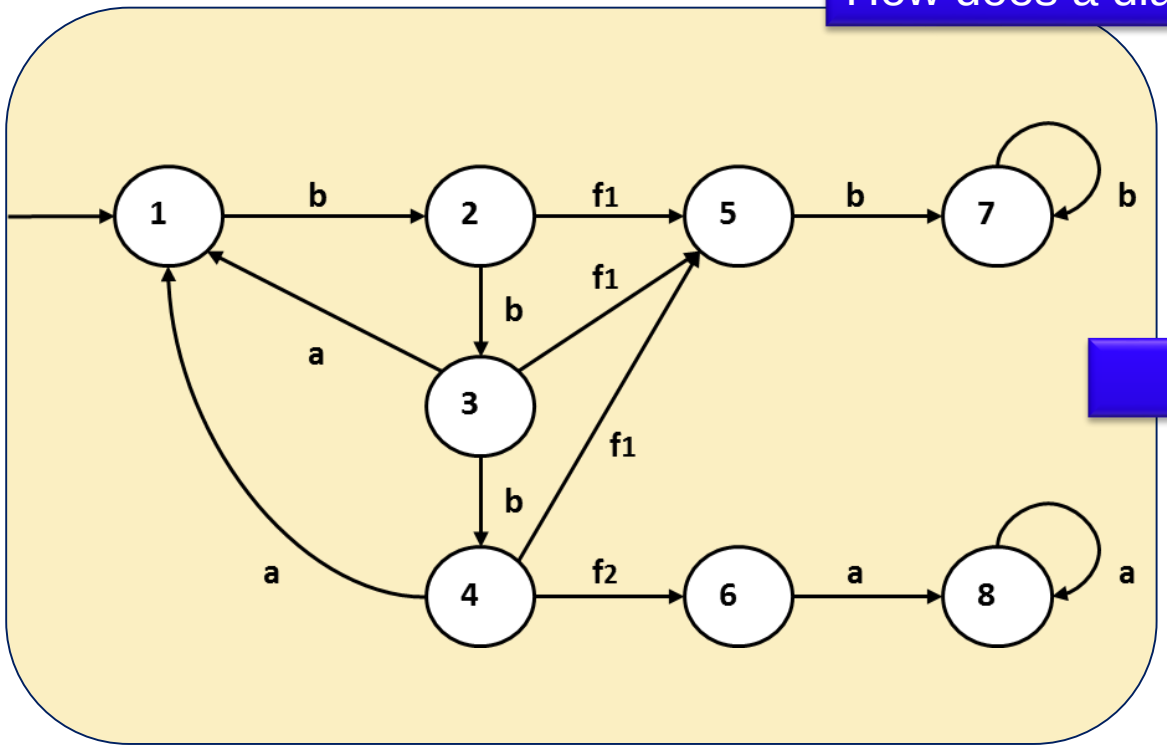
ILLUSTRATIVE EXAMPLE

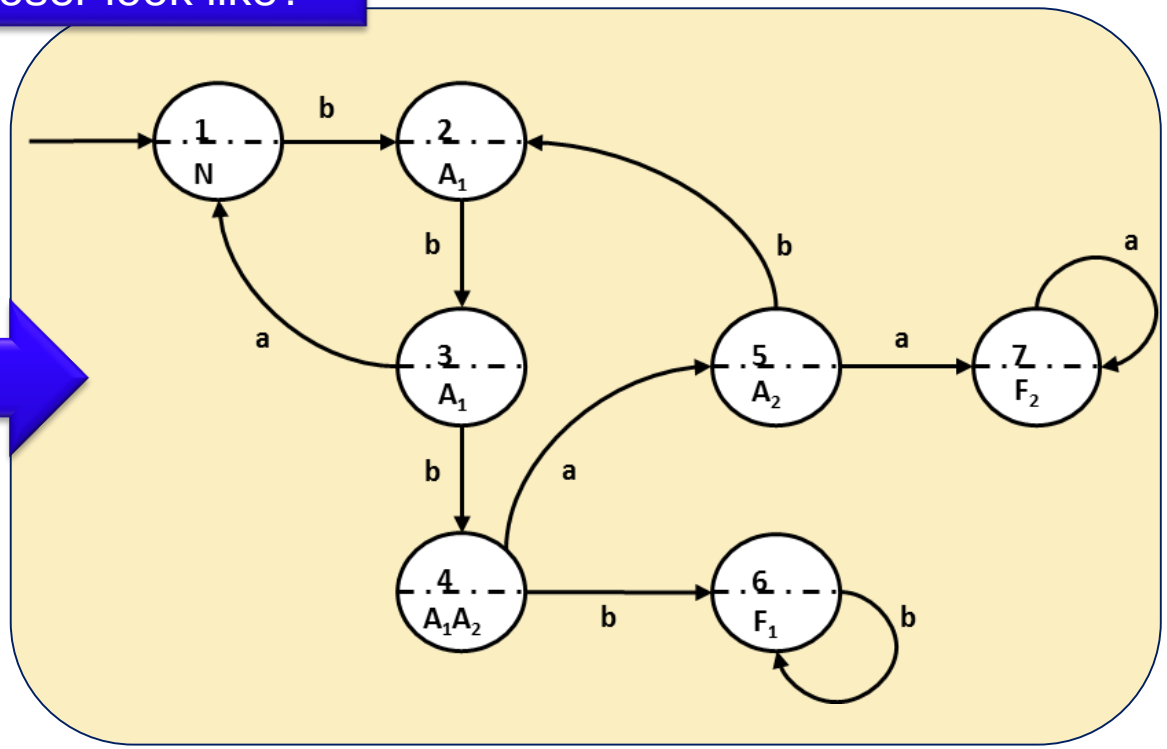
Fire Fighting Autonomous Plane

DES Model for A Fire Fighting Helicopter

The Labeled Automaton Model diagnoser

How does a diagnoser look like?





Question: Assuming we do not have the model of the UAV, how to actively learn and construct the diagnoser?

Construction of the Diagnoser

Role of Oracle

The algorithm constructs the diagnoser G_d by asking minimum queries from an oracle who correctly answers two types of basic queries:

1. Membership queries:

Whether a newly observed string s belongs to $P_{\Sigma o}\,(\mathcal{L}(G))$, and if it is faulty.

2. Equivalence queries:

- Whether $\mathcal{L}(G_d) = P_{\Sigma o}\,(\mathcal{L}(G))$
- If not, the oracle returns a counterexample:

$$cex \in \mathcal{L}(\mathrm{G_d}) \backslash P_{\Sigma \mathrm{o}}\left(\mathcal{L}(G)\right) \cup P_{\Sigma \mathrm{o}}\left(\mathcal{L}(G)\right) \backslash \mathcal{L}(\mathrm{G_d})$$

Construction of the Diagnoser

Observation Table (OT)

The acquired information will be used to create a series of observation tables (S,E,T), where

- $S \subseteq \Sigma^*$ is a non-empty, prefix closed, finite set of strings
- E is a non-empty, suffix closed, finite set of strings
- T is the Condition Map:

$$T(s): (S \cup S.\Sigma_o).E \rightarrow \Delta \cup \{0\}$$

		E
		ε
S	**ε**	N
	b	A_1
$S.\Sigma - S - S_0$	**bb**	A_1

OTs incrementally record and maintain the information about the observed strings.

Construction of the Diagnoser

Condition Map

$$T(\mathrm{w}): (S \cup S.\Sigma_o).E \rightarrow \Delta \cup \{0\}$$

$$T(w) = \begin{cases} \{0\} & \text{if } w \notin P_{\Sigma_o}(\mathcal{L}(G)) \\ \{N\} & \text{if } w \in P_{\Sigma_o}(\mathcal{L}(G)), \text{ and for any } u \in P_{\Sigma_o}^{-1}(w) \rightarrow f_i \notin u \text{ , for } \forall i = 1,2,\dots,n \\ \{L_1, L_2, \dots, L_m\},\ L_i \in \{F_i, A_i\} \\ \quad \bullet\ \{F_i\} \in T(w) & \text{if any } u \in P_{\Sigma_o}^{-1}(w) \text{ contains the failure } f_i. \\ \quad \bullet\ \{A_i\} \in T(w) & \text{if } \exists u, u' \in P_{\Sigma_o}^{-1}(w) \text{ such that } f_i \in u \text{ and } f_i \notin u'. \end{cases}$$

Construction of the Diagnoser

Make it more efficient: Label Propagation

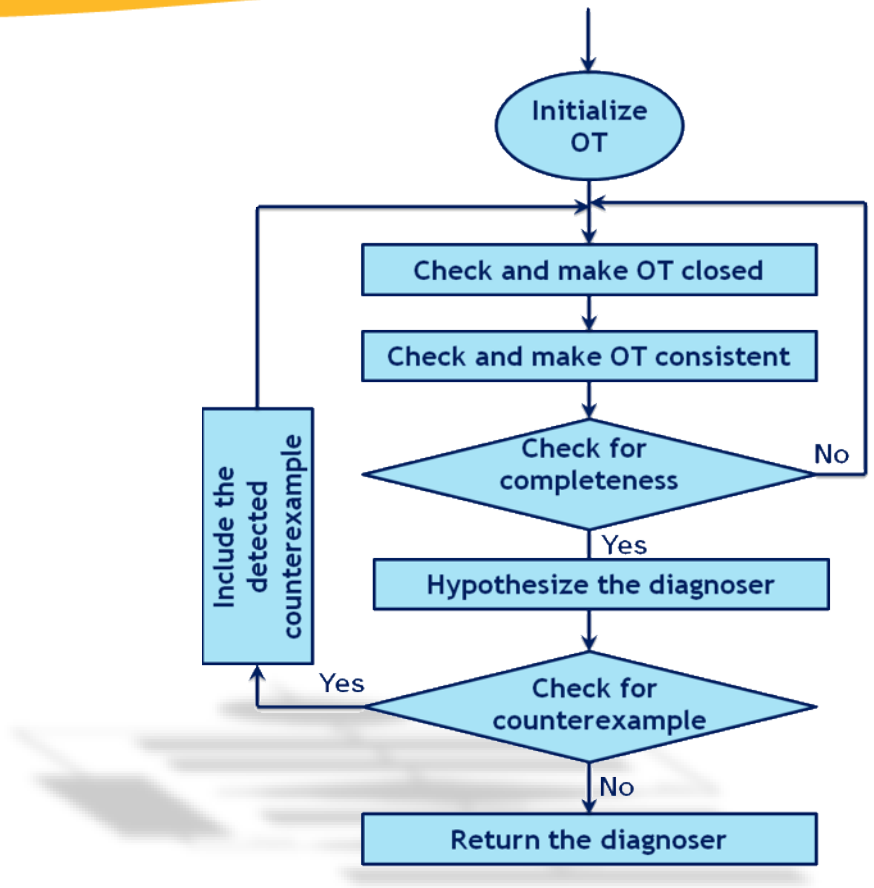
Using this algorithm, some of the queries are possible to be answered using current information in the table:

If a string s is faulty, so are all its possible extensions:

$$[s \in SUS.\Sigma\colon \mathrm{F_i} \in T(s)] \Rightarrow \left[\forall s' \in ext(s) \cap P_{\Sigma o}\left(\mathcal{L}(G)\right)\colon F_i \in T(s')\right]$$

For any string s that is not defined in the system, so are all its extensions:

$$[s \in SUS.\Sigma : \ T(s) = \{0\}] \Rightarrow [\forall s' \in ext(s)\colon T(s') = \{0\}]$$



Algorithm 1 Learning diagnoser algorithm

1: **input:** The observable event set, Σ_o, and the observable language of the system $P_{\Sigma_o}(\mathcal{L}(G))$
2: **output:** The diagnoser G_d with $\mathcal{L}(G_d) = P_{\Sigma_o}(\mathcal{L}(G))$ which is consistent with T
3: **Initialization:** Set S and E to $\{\varepsilon\}$, and form $S.\Sigma$, accordingly.
4: Use the membership queries to build the observation table $T_1(S, E, T)$.
5: Remove $zero-row$ strings from the table T_1.
6: **while** $T_i(S, E, T)$ is not complete **do**
7: **if** T_i is not closed **then**
8: Find $s_1 \in S$ and $\sigma \in \Sigma_o$ such that $row(s_1.\sigma)$ is different from $row(s)$ for all $s \in S$;
9: Add $s_1.\sigma$ to S;
10: Update T_i for $(S \cup S.\Sigma_o).E$ using the label propagation mechanism and membership queries;
11: Remove $zero-row$ strings from $S.\Sigma_o$.
12: **end if**
13: **if** T_i is not consistent **then**
14: Find $s_1, s_2 \in S$, $\sigma \in \Sigma_o$ and $e \in E$ such that $row(s_1) = row(s_2)$ but $T(s_1.\sigma.e) \neq T(s_2.\sigma.e)$;
15: Add $\sigma.e$ to E;
16: Update T_i for $(S \cup S.\Sigma).E$ using the label propagation mechanism and membership queries;
17: Remove zero-row strings from $S.\Sigma$.
18: **end if**
19: **end while**
20: Construct the automaton $G_d(T_i)$ using (9).
21: Ask equivalence query.
22: **if** The oracle replies with the counterexample cex **then**
23: Set $i = i + 1$;
24: Add cex and its prefixes to S;
25: Update T_i for $(S \cup S.\Sigma).E$ using the label propagation mechanism and membership queries;
26: Remove zero-row strings from $S.\Sigma$;
27: Go to $line$ 6.
28: **end if**
29: **return:** $G_d(T_i)$

Section III:

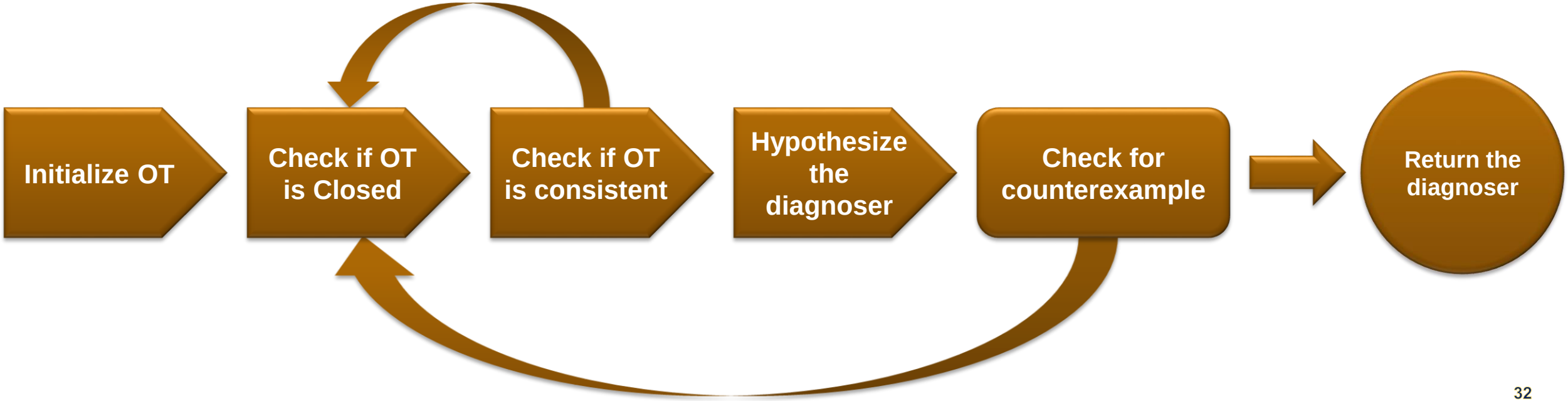
DIAGNOSER FORMATION ALGORITHM

Observation Table (OT)

Construction of the Observation Table (OT)

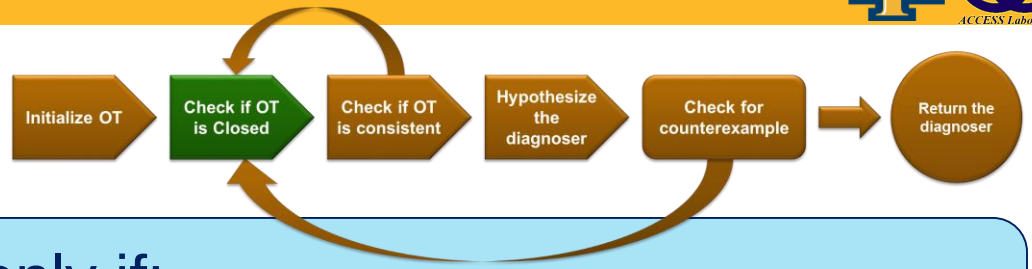
Construction of OT contains Two loops:

- Checking for closeness and consistency
- Checking for any existing counterexample



Observation Table (OT)

Closeness



An observation table is said to be closed if and only if:

$$\forall t \in S.\Sigma_o - S_0, \exists s \in S | \, row(t) = row(s)$$

If the observation table is not closed

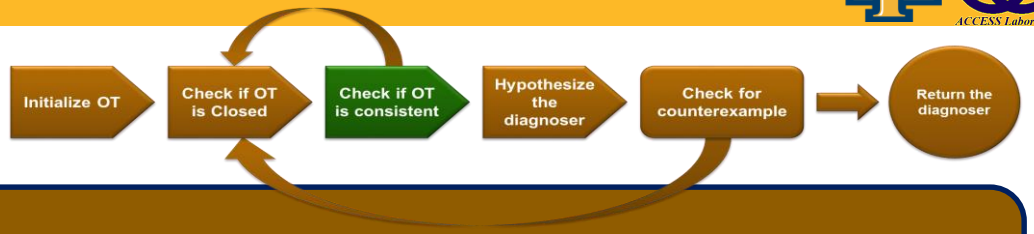
$$\exists s \in S, \exists t \in S.\Sigma_o - S_0 | \, row(t) \neq row(s)$$

To make the observation table closed, add *t* to *S* and update the table.

		E
		ε
S	**ε**	N
$S.\Sigma - S - S_0$	**b**	A_1

		E
		ε
S	**ε**	N
	b	A_1
$S.\Sigma - S - S_0$	**bb**	A_1

Observation Table (OT)



Consistency

An observation table is said to be consistent if and only if:

$$\forall s_1, s_2 \in S \; with \; [row(s_1) = row(s_2)] \Rightarrow [row(s_1.\sigma) = row(s_2.\sigma)], \forall \sigma \in \Sigma_o$$

If the observation table is not consistent,

$$\exists (s_1, s_2) \in S, \exists \sigma \in \Sigma_o, \exists \mathrm{e} \in \mathrm{E} \mid row(s_1) = row(s_2) \; but \; T(s_1 \sigma.\mathrm{e}) \neq T(s_1 \; \sigma.\mathrm{e})$$

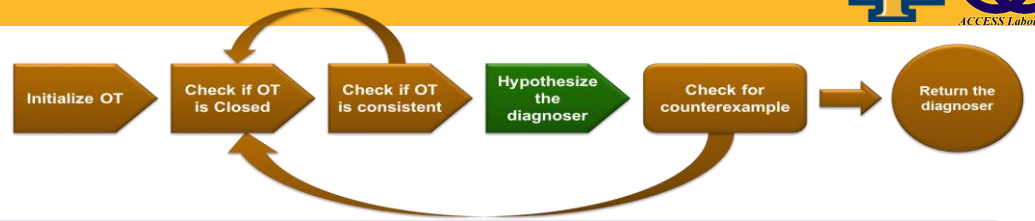
To make the observation table consistent, add $\sigma.e$ to E and update the table.

		E
		ε
S	**ε**	N
	b	A_1
	bb	A_1
	bba	N
	bbb	A_1A_2
$S.\Sigma - S - S_0$	**bbab**	A_1
	bbba	A_2
	bbbb	F_1

		E	
		ε	**a**
S	**ε**	N	0
	b	A_1	0
	bb	A_1	N
	bba	N	0
	bbb	A_1A_2	A_2
$S.\Sigma - S - S_0$	**bbab**	A_1	0
	bbba	A_2	F_2
	bbbb	F_1	0

Observation Table (OT)

Hypothesizing the diagnoser

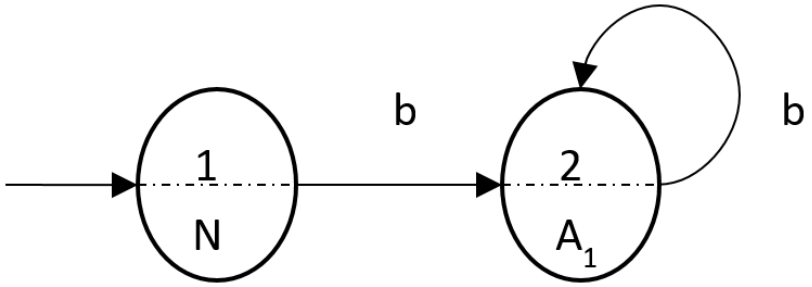


If the observation table is complete (closed and consistent), then we can hypothesize the diagnoser $G_d(T_i)$ based on the observation table OT:

$$G_d(T_i) = (Q_d, \Sigma_d, \Delta_d, \delta_d, h, q_0)$$

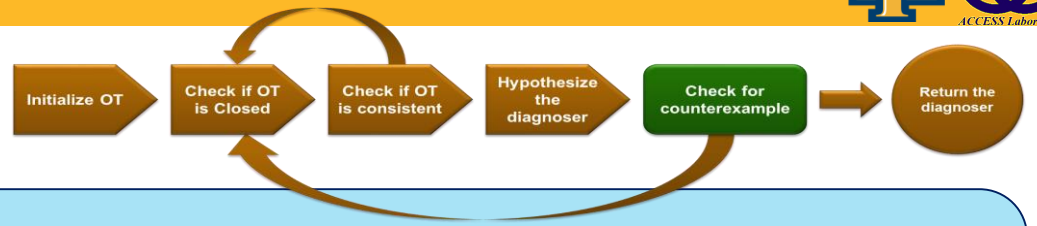
- $\Sigma_d = \Sigma_o$
- $\Delta_p = \Delta \cup \{0\}$
- $Q_d = \{row(s) | s \in S\}$
- $q_0 = row(\varepsilon)$
- $h(row(s)) = T(s.\varepsilon)$
- $\delta_d\,(row(s), \sigma) = row(s.\sigma)$

		E
		ε
S	**ε**	N
	b	A_1
$S.\Sigma - S - S_0$	**bb**	A_1



Observation Table (OT)

Counterexample



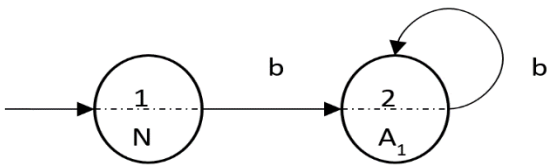
Once the diagnoser was hypothesized, using equivalence queries, the oracle checks for a counter example *cex*:

$$cex \in \mathcal{L}(\mathrm{G_d}) \backslash P_{\Sigma o}\left(\mathcal{L}(G)\right) \cup P_{\Sigma o}\left(\mathcal{L}(G)\right) \backslash \mathcal{L}(\mathrm{G_d})$$

If there exist a counterexample cex:

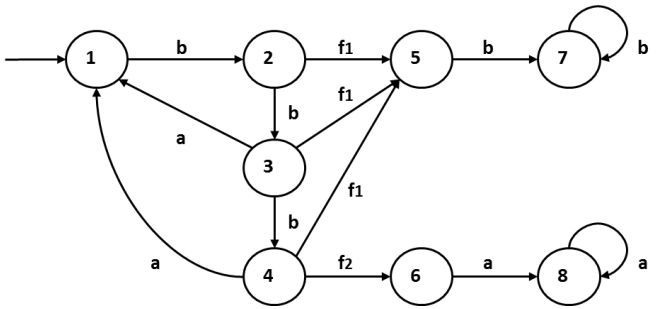
- Add cex and all its prefixes into S, and update table with the new changes.
- This new table again has to be checked for completeness and consistency.

		E
		ε
S	ε	N
	b	A_1
$S.\Sigma - S - S_0$	bb	A_1

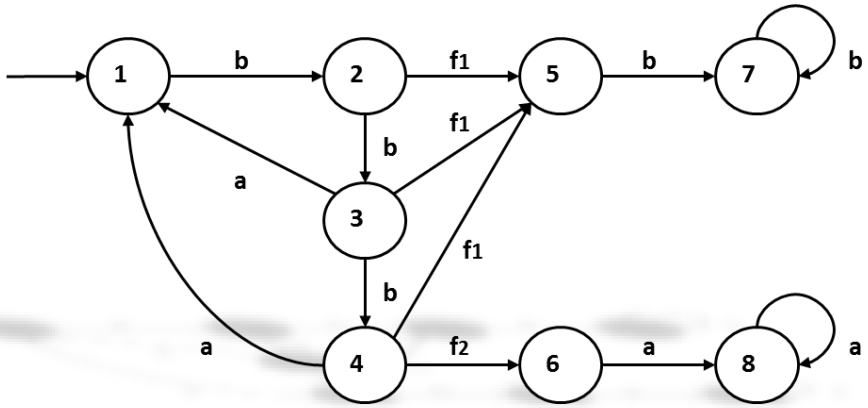


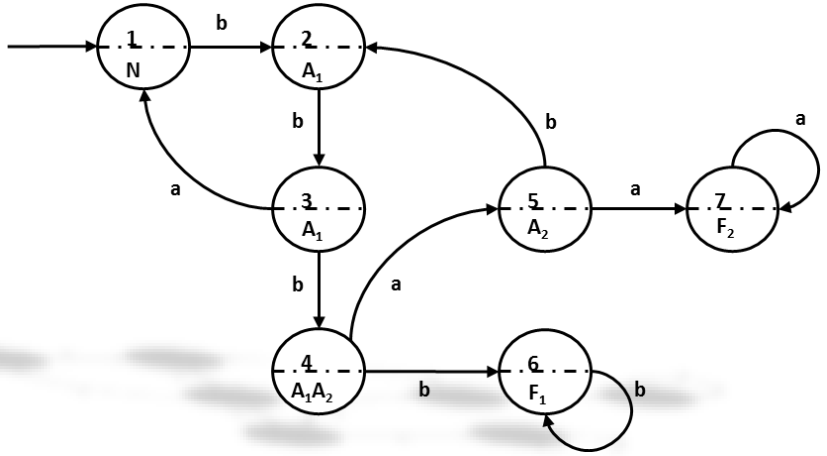
cex= bba

		E
		ε
S	ε	N
	b	A_1
	bb	A_1
	bba	N
$S.\Sigma - S - S_0$	bbb	A_1A_2
	bbab	A_1



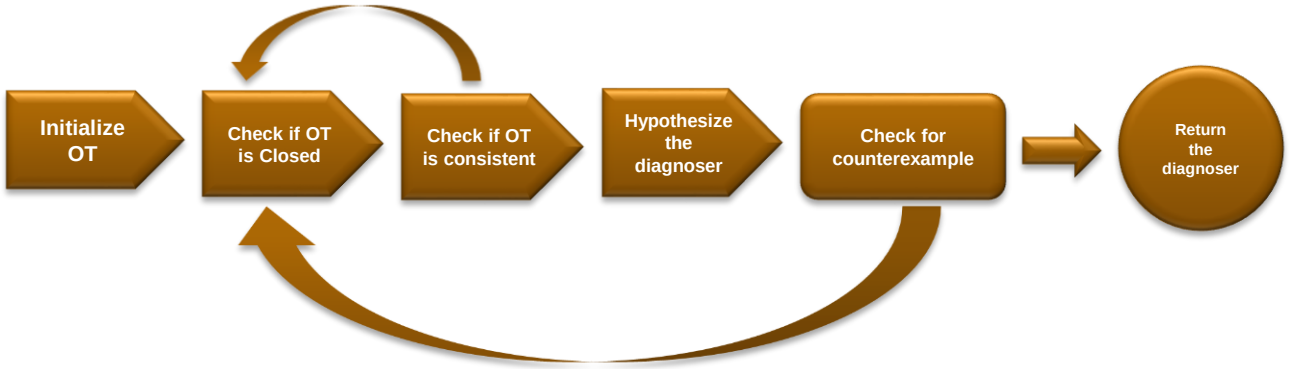
If no counterexample was found, then return the diagnoser.



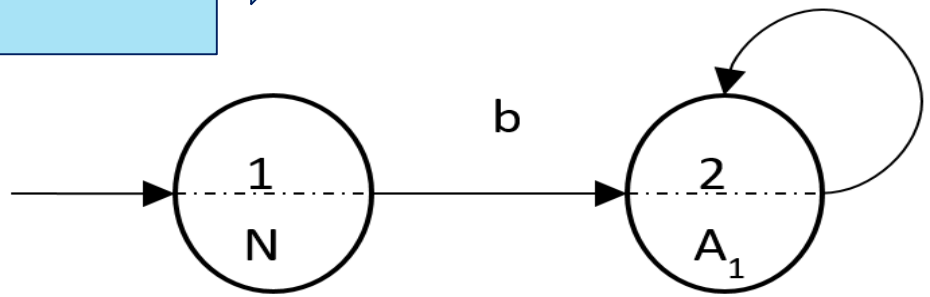


Section IV

ILLUSTRATING EXAMPLE



Initializing OT



OT is not closed

		E
		ε
S	**ε**	N
$S.\Sigma - S - S_0$	**b**	A_1

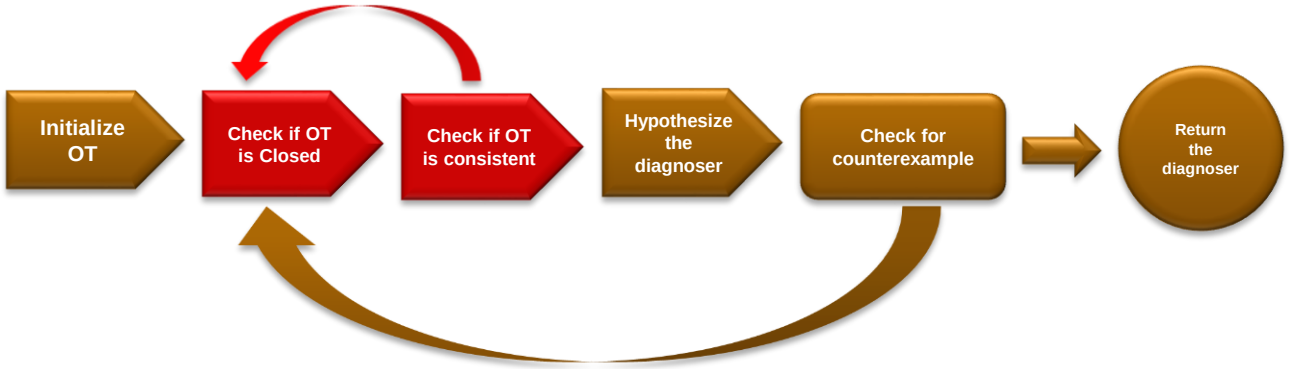
cex= bba

(a)

Hypothesize the diagnoser

		E
		ε
S	**ε**	N
	b	A_1
$S.\Sigma - S - S_0$	**bb**	A_1

(b)



OT is not closed

	E
	ε
ε	N
b	A_1
bb	A_1
bba	N
bbb	A_1A_2
bbab	A_1

S

$S.\Sigma - S - S_0$

(a)

OT is not Consistent

	E
	ε
ε	N
b	A_1
bb	A_1
bba	N
bbb	A_1A_2
bbab	A_1
bbba	A_2
bbbb	F_1

S

$S.\Sigma - S - S_0$

(b)

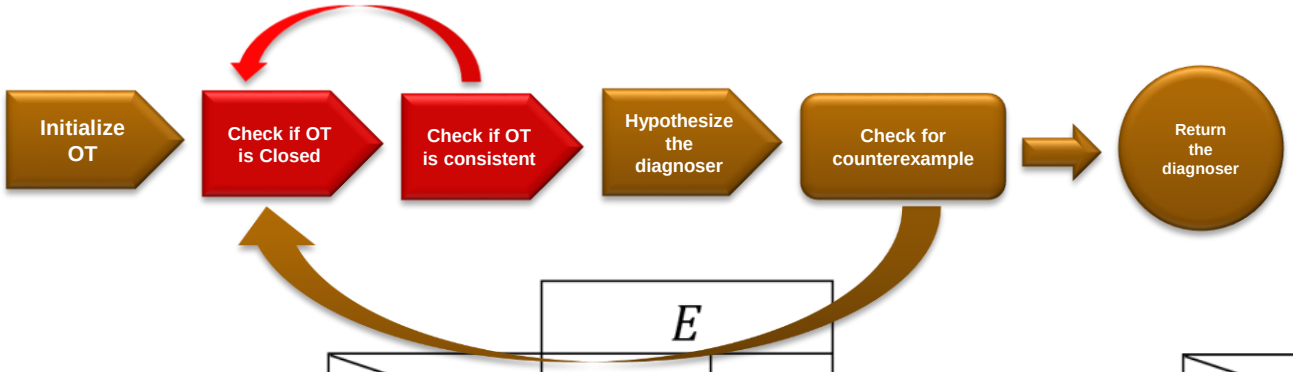
OT is not closed

	E	
	ε	a
ε	N	0
b	A_1	0
bb	A_1	N
bba	N	0
bbb	A_1A_2	A_2
bbab	A_1	0
bbba	A_2	F_2
bbbb	F_1	0

S

$S.\Sigma - S - S_0$

(c)



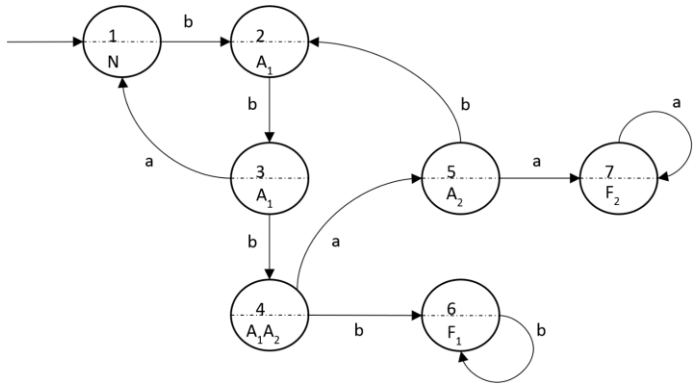
OT is not closed

	E	
	ε	**a**
S	**ε**	N	0
	b	A_1	0
	bb	A_1	N
	bba	N	0
	bbb	A_1A_2	A_2
	bbba	A_2	F_2
$S.\Sigma - S - S_0$	**bbab**	A_1	0
	bbbaa	F_2	F_2
	bbbab	A_1	0
	bbbb	F_1	0

(d)

OT is not closed

	E	
	ε	a
ε	N	0
b	A_1	0
bb	A_1	N
		0
		A_2
		F_2
		0
		0
		F_2
		0
	F_1	0



Return the Diagnoser

Hypothesis the diagnoser

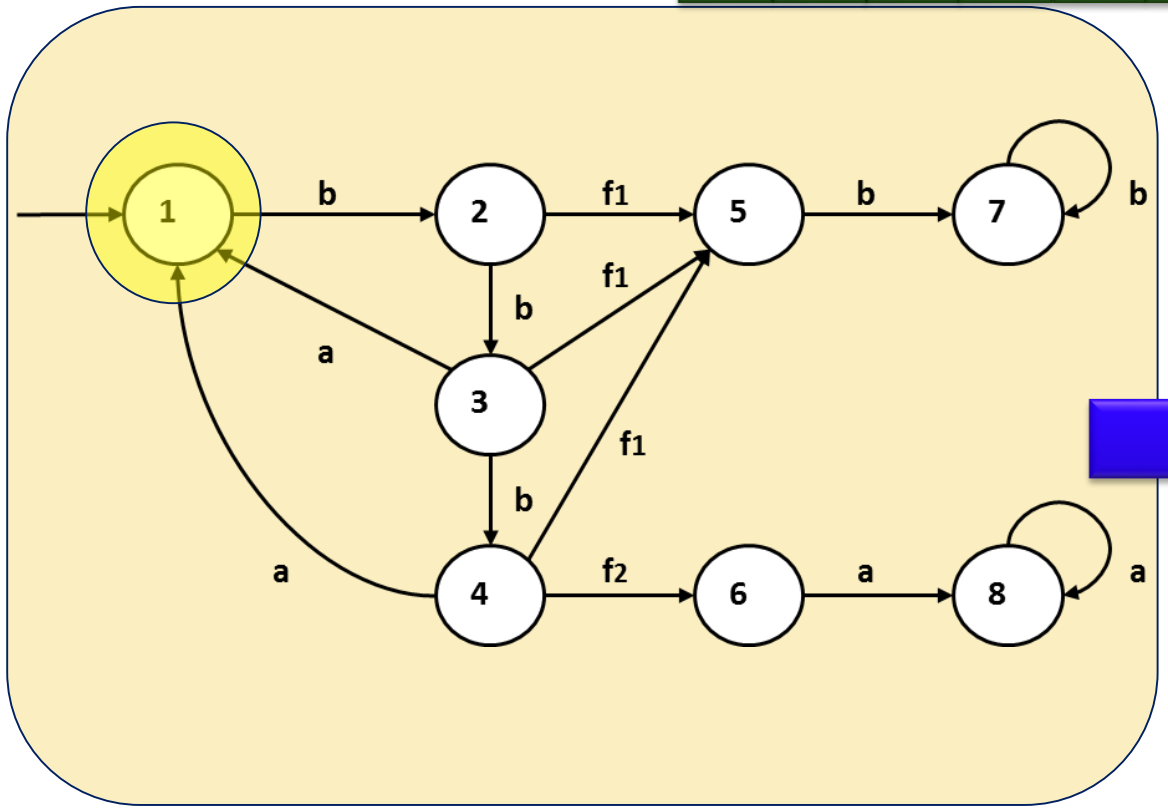
	E	
	ε	**a**
S	**ε**	N	0
	b	A_1	0
	bb	A_1	N
	bba	N	0
	bbb	A_1A_2	A_2
	bbba	A_2	F_2
	bbbb	F_1	0
	bbbaa	F_2	F_2
$S.\Sigma - S - S_0$	**bbab**	A_1	0
	bbbab	A_1	0
	bbbbb	F_1	0
	bbbaaa	F_2	F_2

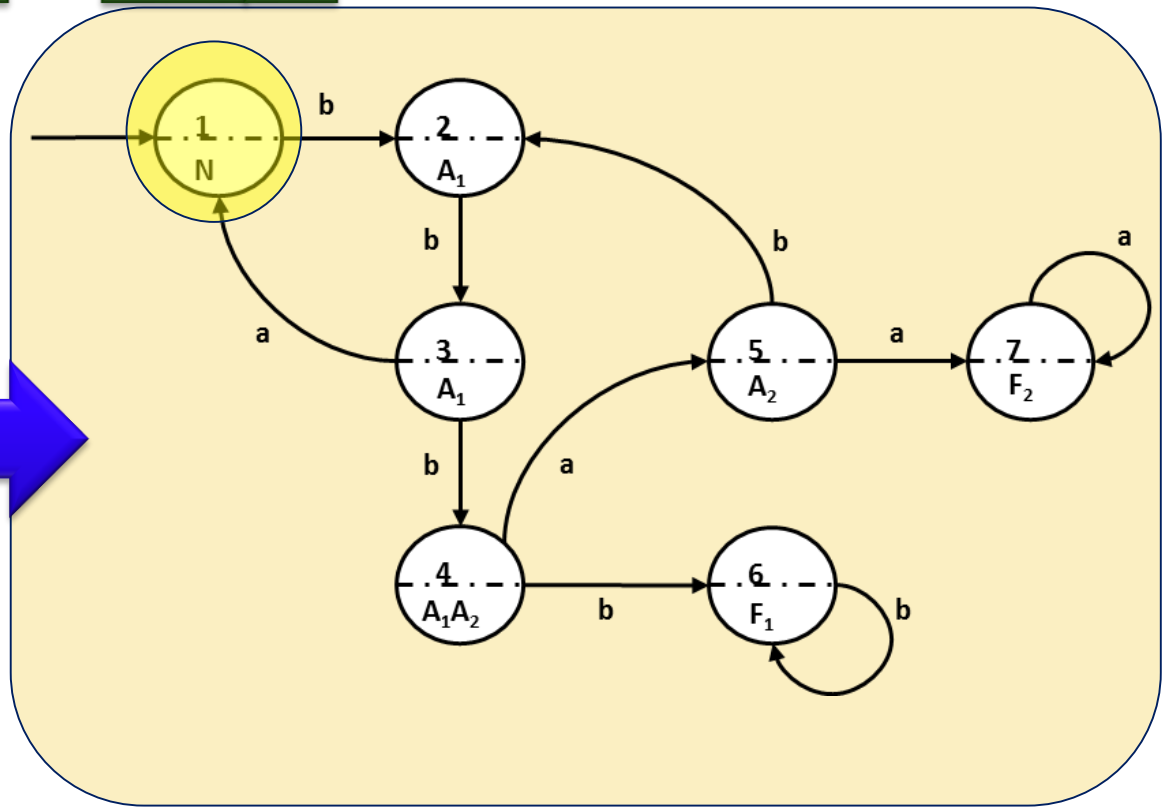
(f)

Online Diagnosing Example

Current Event: b b a b b f_1 b b

Current Label: N A_1 A_1 N A_1 A_1 A_1A_2 F_1





Conclusion

- We discussed the important role of fault diagnosis in making the systems reliable.
- We reviewed existing approaches for fault diagnosis.
- We found Discrete Event system as a capable framework to capture and treat failures in a system
- A novel active-learning approach was proposed for fault diagnosing of an unknown system within DES framework.
- The construction algorithm for the proposed diagnoser was discussed.
- An illustrative example was provided for verifying the algorithm.
- Online implementation of the diagnoser was discussed.

Future Works: Termination of the proposed algorithm

Problem

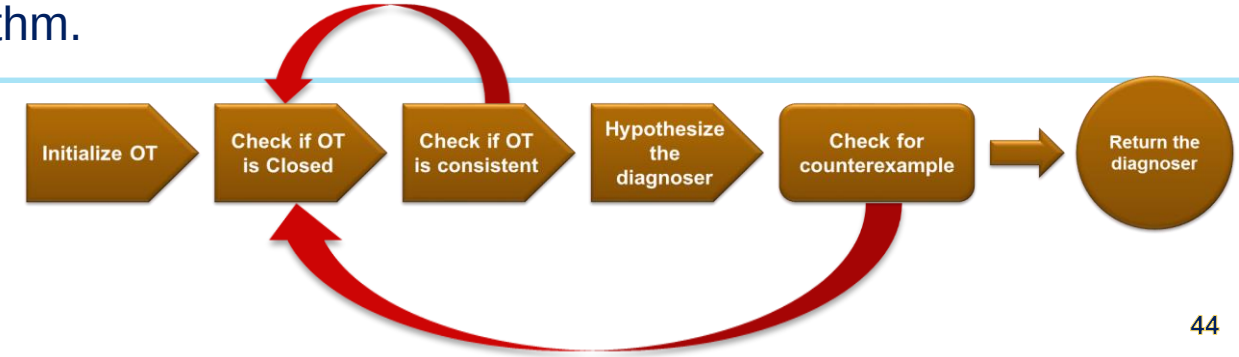
- There two loops in the proposed algorithm. Does the algorithm terminate number of transitions?

Objective

- The proposed algorithm will return the diagnoser in finite time.

Approach

- We will formally prove that both completeness and equivalence loops will terminate after finite number of iterations, so does the whole algorithm.



Future Works: Minimumness of the proposed algorithm

Problem

- Is there any other diagnoser with less number of states that can diagnose the plant? This is important, because the more number of states, the more memory is needed for the implementation of the diagnoser.

Objective

- The proposed algorithm will construct a diagnoser with the minimum number of states.

Approach

- Using lattice theory and regular language theory, we will formally prove that any other diagnoser has equal or more number of states than our proposed diagnoser.

Future Works: Computation cost

Problem

- Complexity of the algorithm can make it un-achievable.

Objective

- We should show be able to provide a standard computational cost value for the proposed algorithm.

Approach

- Using advance analysis of algorithm theories we should be able to calculate the computational cost.

Acknowledgment

- **Dr Ali Karimoddini and my colleagues at ACCESS LAB, particularly Mr. Alejandro White**
- **Financial support from TECHLAV project**

References

- National Aeronautics and Space Administration (NASA). "Nasa Independent Review Team Orb – 3 Accident Investigation Report (Executive Summary)." http://www.nasa.gov/sites/default/files/atoms/files/orb3_irt_execsumm_0.pdf, Oct 19,2015.

- Bregon, Anibal, et al. "An event-based distributed diagnosis framework using structural model decomposition." *Artificial Intelligence* 210 (2014): 1-35.

- Shi, Zhanqun, et al. "The development of an adaptive threshold for model-based fault detection of a nonlinear electro-hydraulic system." *Control Engineering Practice* 13.11 (2005): 1357-1367.

- Leuschen, Martin L., Ian D. Walker, and Joseph R. Cavallaro. "Nonlinear fault detection for hydraulic systems." *Fault Diagnosis and Fault Tolerance for Mechatronic Systems: Recent Advances*. Springer Berlin Heidelberg, 2003. 169-191.

- Velilla, S., and M. Silva. "The spy: A mechanism for safe implementation of highly concurrent systems." *Annual Review in Automatic Programming* 14 (1988): 75-81.

- Silva, Manuel. "Half a century after Carl Adam Petri's Ph. D. thesis: A perspective on the field." *Annual Reviews in Control* 37.2 (2013): 191-219.

References

- Sampath, Meera, et al. "Diagnosability of discrete-event systems." *Automatic Control, IEEE Transactions on* 40.9 (1995): 1555-1575.
- Dai, Xuewu, and Zhiwei Gao. "From model, signal to knowledge: A data-driven perspective of fault detection and diagnosis." *Industrial Informatics, IEEE Transactions on* 9.4 (2013): 2226-2238.
- Dong, Hongli, et al. "Fuzzy-model-based robust fault detection with stochastic mixed time delays and successive packet dropouts." *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on* 42.2 (2012): 365-376.
- Y. Zheng, H. Fang, and H. O. Wang, “Takagi-sugeno fuzzy-modelbased fault detection for networked control systems with markov delays,” Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, vol. 36, no. 4, pp. 924–929, 2006.
- Wu, Jianing, Shaoze Yan, and Liyang Xie. "Reliability analysis method of a solar array by using fault tree analysis and fuzzy reasoning Petri net." *Acta Astronautica* 69.11 (2011): 960-968.
- W. Lee, D. Grosh, F. Tillman, and C. Lie, “Fault tree analysis, methods, and applications,” Reliability, IEEE Transactions on, vol. R-34, no. 3, pp. 194–203, Aug 1985.
- Zhang, Pinjia, et al. "A survey of condition monitoring and protection methods for medium-voltage induction motors." *Industry Applications, IEEE Transactions on* 47.1 (2011): 34-46.
- Angluin, Dana. "Learning regular sets from queries and counterexamples."*Information and computation* 75.2 (1987): 87-106.

Q&A

Thank you

MohammadMahdi Karimi

mmkarimi@aggies.ncat.edu

TECHLAV

ACCESS Laboratory

Our Website:
techlav.ncat.edu
accesslab.net