Fault Diagnosis for an Unknown Plant

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

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

Fault Diagnosis for an Unknown Plant
MohammadMahdi Karimi

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))*

**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

**MOTIVATION**

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.
Outline

1. Introduction to Fault Diagnosing
2. Learning Diagnoser for and Unknown Plant
3. Diagnoser Construction Algorithm
4. Illustrative Example
5. Future Work

Title: Fault Diagnosis for an Unknown Plant
Author: MohammadMahdi Karimi
Facts:
- Despite all our efforts, failures in the system cannot be avoided.
- Failures may happen everywhere 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.
**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”.

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.

Background Review

Fault Diagnosis

Model Based

Time Driven

- Linear Models
  - Shi, Zhanqun, et al. (2005)
- Nonlinear Models
  - L. Leuschen et al. (2003)
- Petri net
  - M. Silva (2013)
  - Velilla & Silva (1988)
  - A. Bregon, et al. (2014)
  - Sampath et al. (1995)

Event Driven

- Automata
  - Neural Network
    - Dai, Xuewu, et al. (2013)
  - Fuzzy
    - Dong, Hongli, et al. (2012)
    - Y. Zheng et al. (2006)
  - Fault Tree
    - Wu, Jianing, et al. (2011)
    - W. Lee et al. (1985)

Model Free

Data Driven

Analytical Approaches

- Spectrum Analysis
  - Zhang, Pinjia, et al. (2011)

Passive Learning
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 computational methods where more detailed formal methods become necessary for their analysis and design. Indeed, the use of DESs 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 [20], [37] and manufacturing systems [36]; and, more recently, has been extended to cover modular [36], [39] and distributed [9], [33], [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 synthesis problems.

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

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

Example:

\[
G = (X, \Sigma, \delta, x_0, X_m)
\]

- \(X\): state space
- \(\Sigma\): event set
- \(\delta\): partial transition
- \(x_0\): initial state
- \(X_m\): final states

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

\[\delta(\text{Hangar Mode}, S) = \text{Mission Mode}\]
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.

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.
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.
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
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 $x \rightarrow x'$) 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 $P_{\Sigma_0} (L(G))$. 
Properties of DES Diagnoser

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

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

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

Output label set:

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

- $N$: normal
- $F_i$: occurrence of the failure $f_i$
- $A_i$: ambiguity in the occurrence of the failure $f_i$
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:
     
     $$cex \in \mathcal{L}(G_d) \setminus P_{\Sigma_o}(\mathcal{L}(G)) \cup P_{\Sigma_o}(\mathcal{L}(G)) \setminus \mathcal{L}(G_d)$$
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:
  \[ T(s): (S \cup S_0) \cdot E \to \Delta \cup \{0\} \]

OTs incrementally record and maintain the information about the observed strings.
Construction of the Diagnoser

Condition Map

\[
T(w) = \begin{cases} 
\{0\} & \text{if } w \not\in P_{\Sigma_0}(L(G)), \\
\{N\} & \text{if } w \in P_{\Sigma_0}(L(G)), \text{ and for any } u \in P_{\Sigma_0}^{-1}(w) \rightarrow f_i \not\in u, \text{ for } \forall i = 1, 2, ..., n, \\
\{L_1, L_2, ..., L_m\}, L_i \in \{F_i, A_i\} & \text{if any } u \in P_{\Sigma_0}^{-1}(w) \text{ contains the failure } f_i, \\
\{F_i\} \in T(w) & \text{if } \exists u, u' \in P_{\Sigma_0}^{-1}(w) \text{ such that } f_i \in u \text{ and } f_i \not\in u'. 
\end{cases}
\]
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:

$$[s \in \text{SUS. } \Sigma: F_i \in T(s)] \Rightarrow [\forall s' \in \text{ext}(s) \cap P_{\Sigma_0}(L(G)): F_i \in T(s')]$$

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

$$[s \in \text{SUS. } \Sigma: T(s) = \{0\}] \Rightarrow [\forall s' \in \text{ext}(s): T(s') = \{0\}]$$
Algorithm 1 Learning diagnoser algorithm

1. **Input**: The observable event set, $\Sigma_o$, and the observable language of the system $P_{\Sigma_o}(L(G))$
2. **Output**: The diagnoser $G_d$ with $L(G_d) = P_{\Sigma_o}(L(G))$
3. **Initialization**: Set $S$ and $E$ to $\{e\}$, and form $S.S$, 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 $T_1(S,E,T)$ is not complete do
   - If $T_1$ is not closed then
     - Find $s_1 \in S$ and $\sigma \in \Sigma_o$ such that $\text{row}(s_1,\sigma)$ is different from $\text{row}(s)$ for all $s \in S$;
     - Add $s_1,\sigma$ to $S$;
     - Update $T_1$ for $(S \cup S.S,S,E)$ using the label propagation mechanism and membership queries;
     - Remove zero-row strings from $S.S$.
   - End if
   - If $T_1$ is not consistent then
     - Find $s_1, s_2 \in S, \sigma \in \Sigma_o$ and $e \in E$ such that $\text{row}(s_1) = \text{row}(s_2)$ but $T(s_1,\sigma,e) \neq T(s_2,\sigma,e)$;
     - Add $\sigma,e$ to $E$;
     - Update $T_1$ for $(S \cup S.S,E)$ using the label propagation mechanism and membership queries;
     - Remove zero-row strings from $S.S$.
   - End if
7. End while
8. Construct the automaton $G_d(T_1)$ using (9).
10. If the oracle replies with the counterexample $cex$ then
11.   - Set $i = i + 1$;
12.   - Add $cex$ and its prefixes to $S$;
13.   - Update $T_i$ for $(S \cup S.S,E)$ using the label propagation mechanism and membership queries;
14.   - Remove zero-row strings from $S.S$;
15.   - Go to line 6.
16. End if
17. Return $G_d(T_i)$.
Construction of the Observation Table (OT)

Construction of OT contains Two loops:

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

Initialization of OT → Check if OT is Closed → Check if OT is consistent → Hypothesize the diagnoser → Check for counterexample → Return the diagnoser
Closeness

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

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

If the observation table is not closed

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

To make the observation table closed, add $t$ to $S$ and update the table.
An observation table is said to be consistent if and only if:

\[ \forall s_1, s_2 \in S \text{ with } \text{row}(s_1) = \text{row}(s_2) \Rightarrow \text{row}(s_1 \cdot \sigma) = \text{row}(s_2 \cdot \sigma), \forall \sigma \in \Sigma_0 \]

If the observation table is not consistent,

\[ \exists (s_1, s_2) \in S, \exists \sigma \in \Sigma_0, \exists e \in E \mid \text{row}(s_1) = \text{row}(s_2) \text{ but } T(s_1 \cdot \sigma \cdot e) \neq T(s_1 \cdot e) \]

To make the observation table consistent, add \( \sigma \cdot e \) to \( E \) and update the table.
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:

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

- $\Sigma_d = \Sigma_o$
- $\Delta_p = \Delta \cup \{0\}$
- $Q_d = \{\text{row}(s) | s \in S\}$
- $q_0 = \text{row}(\epsilon)$
- $h(\text{row}(s)) = T(s.\epsilon)$
- $\delta_d (\text{row}(s), \sigma) = \text{row}(s.\sigma)$
Once the diagnoser was hypothesized, using equivalence queries, the oracle checks for a counter example \( cex \):

\[
\text{\( cex \in \mathcal{L}(G_d) \setminus P_{\Sigma_0} (\mathcal{L}(G)) \cup P_{\Sigma_0} (\mathcal{L}(G)) \setminus \mathcal{L}(G_d) \)}
\]

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.
Section IV

ILLUSTRATING EXAMPLE
Fault Diagnosis for an Unknown Plant

MohammadMahdi Karimi

Initializing OT

OT is not closed

Hypothesize the diagnoser

(a) cex = bba

(b)

<table>
<thead>
<tr>
<th>S</th>
<th>ε</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>A₁</td>
<td></td>
</tr>
</tbody>
</table>

S. Σ - S - S₀

<table>
<thead>
<tr>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
</tr>
<tr>
<td>N</td>
</tr>
</tbody>
</table>

S. Σ - S - S₀

<table>
<thead>
<tr>
<th>b</th>
<th>A₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>bb</td>
<td>A₁</td>
</tr>
</tbody>
</table>
Fault Diagnosis for an Unknown Plant

Mohammad Mahdi Karimi

OT is not closed

\[ S \]

\[ E \]

\[ \varepsilon \]

\[ N \]

\[ b \]

\[ A_1 \]

\[ bb \]

\[ A_1 \]

\[ bba \]

\[ N \]

\[ S. \Sigma - S - S_0 \]

\[ bbb \]

\[ A_1 A_2 \]

\[ bbab \]

\[ A_1 \]

(b) OT is not consistent

\[ E \]

\[ \varepsilon \]

\[ N \]

\[ b \]

\[ A_1 \]

\[ bb \]

\[ A_1 \]

\[ bba \]

\[ N \]

\[ bba \]

\[ N \]

\[ bba \]

\[ N \]

\[ bba \]

\[ N \]

\[ bba \]

\[ N \]

\[ bbab \]

\[ A_1 \]

\[ bbba \]

\[ A_2 \]

\[ bbbb \]

\[ F_1 \]

(c) OT is not closed

\[ E \]

\[ \varepsilon \]

\[ a \]

\[ \varepsilon \]

\[ N \]

\[ 0 \]

\[ b \]

\[ A_1 \]

\[ 0 \]

\[ bb \]

\[ A_1 \]

\[ N \]

\[ bba \]

\[ N \]

\[ 0 \]

\[ bba \]

\[ N \]

\[ 0 \]

\[ bbab \]

\[ A_1 \]

\[ 0 \]

\[ bbba \]

\[ A_2 \]

\[ F_2 \]

\[ bbbb \]

\[ F_1 \]

\[ 0 \]
Fault Diagnosis for an Unknown Plant

Mohammad Mahdi Karimi

OT is not closed

Hypothesis the diagnoser

Initialize OT

Check if OT is Closed

Check if OT is consistent

Check for counterexample

Return the diagnoser

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & N & 0 \\
bbba & A_1A_2 & A_2 \\
bbba & A_2 & F_2 \\
bbab & A_1 & 0 \\
bbba & F_2 & F_2 \\
bbbab & A_1 & 0 \\
bbbb & F_1 & 0 \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
b & A_1 & 0 \\
bb & A_1 & N \\
bbba & A_1 & N \\
\end{array}
\]

\[
\begin{array}{ccc}
\epsilon & a \\
\epsilon & N & 0 \\
Online Diagnosing Example

Current Event: [b b a b b f₁ b b]

Current Label: [N A₁ A₁ N A₁ A₁ A₁ A₂ F₁]
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 are two loops in the proposed algorithm. Does the algorithm terminate after a finite 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 a 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-achieviable.

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


References


Thank you

Mohammad Mahdi Karimi
mmkarimi@aggies.ncat.edu

Merry Christmas