Semi-asynchronous Fault Diagnosis of Discrete Event Systems

ALEJANDRO WHITE
DR. ALI KARIMODDINI

OCTOBER 2017

NC A&T State University
http://www.ncat.edu/

http://techlav.ncat.edu/

ACCESS Laboratory
http://accesslab.net/
Why Do We Need Fault Diagnosis?

As systems have become larger, more complex, and more integrated into our daily lives, it is imperative and obligatory that there exists systematic fault diagnosis techniques that provide a timely and accurate diagnosis of system behaviors.
Increase of roles and presence in daily societal activities leads to an increase in liability.

Crash Report

Google’s self-driving car caused its first crash.

Perrone Robotics driverless car crashes after being hacked during testing.
Failure to properly diagnose faults, leads to incorrect recovery actions

American Airlines Flight 191 (1979)
- Left Engine separated from wing
- Pilot only 15s to react
- Subsequent analysis shows consequence of faults avoidable

“Wow, pulled back the wrong side throttle” ...... (captain of TransAsia Flight GE235)
### General Objectives & Impacts of Fault Diagnosis

<table>
<thead>
<tr>
<th>Objectives</th>
<th>Impacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>To develop systematic techniques for <strong>automatic diagnosis</strong> of faults in the system to <strong>timely</strong> diagnose (detect, identify and locate) occurred system faults.</td>
<td>Upon a fault occurrence, a system will autonomously become aware of the fault’s occurrence, and initiate a systematic procedure that locates, identifies, and accommodates the fault in order to ensure proper utilization of the system’s remaining resources, allowing for a resilient post fault system operation that is both safe and stable.</td>
</tr>
</tbody>
</table>
Why Discrete Event System (DES)?

**Behavior Model**
The arbitrary nature of unobservable fault occurrences leads to the non-deterministic, non-linear behavior of highly complex systems; inherently making a discrete-event modelled system representation quite suitable for diagnosing fault occurrences.

**Cause & Effect**
The common structure of DES consists of various sequences of events/actions leading to various system states. This structure matches a human’s instinctive perception of cause and effect, thus providing for more natural intuitive system analytics.

**Topology**
The topology of a DES, represents a system’s behavior as sequences of discrete events. This allows for the capturing of disruptive changes in a system’s operation; in turn highlighting faulty behaviors of the system.

Alejandro White  
Semi-asynchronous Fault Diagnosis of Discrete Event Systems
Preliminaries and Background
Automaton

- **Definition**: a non-deterministic finite-state Discrete-Event System (DES) can be represented by a four-tuple: $G = (X, \Sigma, \delta, x_0)$

- **State space** ($X$): a discrete set of system states

- **Event set** ($\Sigma = \Sigma_o \cup \Sigma_u$): notable occurrences of asynchronous discrete changes in a system
  - **Observable events** ($\Sigma_o$): events observed by a sensor (e.g., flowing of water)
  - **Unobservable events** ($\Sigma_u$): events that are unable to be detected by sensors; possibly due to sensor absence/damage (e.g., failure event)

- **State-transition relation** ($\delta: X \times \Sigma \rightarrow 2^X$): a partial relation that determines all feasible system state transitions caused by system events ($2^X$ is the set of all possible combinations of states)

- **Initial state** ($x_0$): indicated by an input arrow connected to a single state

\[\begin{align*}
\delta(1, u) &= 2, \quad \delta(2, \alpha) = \{2, 3\}, \\
\delta(3, \beta) &= 3
\end{align*}\]

\[x_0 = \{1\}\]
Automaton (Language)

- **Definition:** The system language is a discrete representation of the system’s behaviors (normal and faulty) in the form of sequences of events.

- **Trace (string):** A sequence of one or more events, allowable by the system’s behavior.
  - E.g., \( s = e_1 e_2 \ldots e_n \) where \( e_i \in \Sigma \)

- **Language \( \mathcal{L}_G(x_0) \):** The set of all system traces which originate at the system’s initial state \( x_0 \)
  - \( \mathcal{L}_G(x_0) = \{ s \in \Sigma^* | \delta(x_0, s) \text{ is defined} \} \)
  - \( \Sigma^* \) is the Kleene Closure of \( \Sigma \)

\[ \mathcal{L}_G(x_0) = \{ \epsilon, u, u\alpha^*, u\alpha^*\alpha, u\alpha^*\beta^*, u\alpha^*\beta\beta^*, \ldots \} \]

\[ \mathcal{L}_G(x_0) = \{ \epsilon, u\alpha^*\beta^* \} \]

*: arbitrarily repeated string
Our purpose is to diagnose unobservable faults from the observable behavior of the system.

The system’s observable behavior can be described by the natural projection ($P$) of the system’s language to the observable language set of the system.

\[ P: \Sigma^* \rightarrow \Sigma_0^* \]

- \[ P(\varepsilon) = \varepsilon \]
- \[ P(e) = e \quad \text{if } e \in \Sigma_0 \]
- \[ P(e) = \varepsilon \quad \text{if } e \notin \Sigma_0 \]
- \[ P(se) = P(s)P(e) \quad \text{for } s \in \Sigma^* \text{ and } e \in \Sigma \]

Extension of the natural projection to the languages:

\[ P(\mathcal{L}_G(x_0)) = \{P(s) \mid s \in \mathcal{L}_G(x_0)\} \]

Inverse of Natural Projection

\[ P^{-1}_{\mathcal{L}_G(x_0)}(w) = \{s \in \mathcal{L}_G(x_0) \mid P(s) = w\} \]

\[ \mathcal{L}_G(x_0) = \{u, ua^*\beta^*\} \]

\[ P(\mathcal{L}_G(x_0)) = \{\varepsilon, \alpha^*\beta^*\} \]

\[ P^{-1}_{\mathcal{L}_G(x_0)}(\alpha) = \{ua\} \]
Here we present pre-defined sets of system strings

**Extension Closure:** \( \text{ext}(\mathcal{L}_G(x_0)) := \{v \in \Sigma^* \mid \exists u \in \mathcal{L}_G(x_0): uv \in \mathcal{L}_G(x_0)\} \)

**Pre\(X_s\):** \(\text{Pre}(X_s) = \{s \in \mathcal{L}_G(x_0) : \delta(x_0, s) \in X_s\}\), the set of strings leading to \(X_s\) generated from \(x_0\)

**Post\(X_s\):** the set of strings that can be generated from any \(x \in X_s\)

**Unobservable Reach:** \(UR(s) = \{y \in X \mid \exists u \in \Sigma_u^*, \delta(x, u) = y\}\), the set of all system states (with the inclusion of \(x\) itself) reachable from state \(x\) via strings solely consisting of unobservable events

**Unobservable Extension:** \(UE(s, x) = \{s.t \mid t \in \Sigma_u^* \text{ and } s.t \in \mathcal{L}_G(x)\}\), specifies the set of all unobservable extensions of \(s\) concatenated with the string \(s\), and generated from the state \(x\)
Fault diagnostics is provided by the diagnoser.

The diagnoser extracts information from the original system’s observable behaviors, in order to estimate the original system’s current state and current condition (faulty or non-faulty).

The diagnoser’s transitions are only defined over the original system’s observable event occurrences.

Upon observance of the original system’s behavior, the diagnoser updates its estimation of the original system’s state and condition.
DES Fault Diagnosis (State-Of-The-Art)

**STRUCTURE**
- Centralized: Sampath et al. 1995, Zad et al. 2003
- Decentralized: Wang et al. 2007, Lafortune et al. 2005
- Distributed: Fabre 2002, Pencolé 2005

**STRATEGY**
- Event-based: Sampath et al. 1995

**TOOLS**
- Process Algebra: Console et al. 2002
- Automata Theory: Sampath 1995, Wang et al., 2007
In many practical situations, only after a fault occurs, the diagnosis tool can be brought in and connected to the faulty plant to diagnose the occurred fault.

In all of existing methods, the diagnoser has to be simultaneously initialized with the system under diagnosis.

In many cases, it is not possible, or it is time-consuming and costly, to restart the plant to be synchronized with the diagnoser.

In all of existing methods, the diagnoser should synchronously execute the events in parallel with system under diagnosis, to keep the past history of exhibited normal and faulty behaviors.
Objectives of This Research

- To develop automatic diagnosis techniques to timely diagnose (detect, identify and locate) occurred faults.
- To develop a diagnosis approach that can definitively diagnose all modelled system fault occurrences.
- To construct a semi-asynchronous DES fault diagnoser, which is not required to be synchronously initialized with the system under diagnosis (i.e., it can work without requiring the restarting of the system).
Challenges

I. Unlike conventional diagnosis techniques, the past history of the system before the activation of the semi-asynchronous diagnoser is not available, leaving the semi-asynchronous diagnoser with the challenge of diagnosing faults using only the future behaviors of the plant observed after the activation of the semi-asynchronous diagnoser.

II. In contrast to existing methods, where the initial state of the system and correspondingly the initial state of the diagnoser are generally assumed to be non-faulty; upon its initialization, the semi-asynchronous diagnoser is not able to assume that the current condition of the system is normal.
Contributions

- Introduced formal definition of $F_i$-Semi-Asynchronous Diagnosability
- Algorithm for constructing Semi-Asynchronous Diagnoser
- Novel diagnoser capable of diagnosing multiple-typed system faults without the commonplace requirement of system reinitialization
- Model-based, and definition-based verification of $F_i$-Semi-Asynchronous Diagnosability

DES Fault Diagnosis
When there is uncertain information about the state of the system upon activation of the diagnoser, how do we distinctively characterize a system’s behavior (system state and condition) solely based upon a finite number of subsequent successive external system observations.
Problem Formulation

Consider the DES plant $G = (X, \Sigma, \delta, x_0)$ where $X, \Sigma, \delta$ and $x_0$ represent the system’s state space, event set, state transition relation, and the system’s initial state, respectively.

Consider that the event set $\Sigma = \Sigma_o \cup \Sigma_u$ can be disjointly partitioned into two subsets: $\Sigma_o$ (the observable event set) and $\Sigma_u$ (the unobservable event set which consists of unobservable events including faults $\Sigma_f \subseteq \Sigma_u$).

Consider $X_s$ as the initial estimation of possible state locations of plant $G$, upon the activation of the diagnoser.

Then

- For any string $s$ that belongs to the language of the plant ($s \in \mathcal{L}_G(x_0)$) that leads to one of the states in $X_s$ ($\delta(x_0, s) \in X_s$).

- And for any of its sufficiently long suffix strings $t$ ($t \in \mathcal{L}_{G/s}$) where $t$ occurs after the diagnoser activation.

Determine from the observable behavior of the system, $P(t)$

- If a fault has occurred (i.e., check if $\exists f \in \Sigma_f$ such that $f \in s \cdot t$)
- If yes, determine the type of the fault $\Sigma_{f_i}$ where $f \in \Sigma_{f_i}$
- Locate the system state $x \in X$ that is subsequently reached by $s \cdot t$
Proposed Solution

We propose a novel algorithm that generates a deterministic finite state event-based diagnoser capable of diagnosing multiple, non-permanent system faults without requiring the restarting of the system under diagnosis.

The algorithm produced semi-asynchronous diagnoser, starts with an uncertain estimation of the system under diagnosis, and upon gathering run-time system observations, the diagnoser refines its estimates of the original system’s state and condition.

Diagnoser capable of diagnosing system faults the occur before and after diagnoser activation.

Our design takes advantage of the fact that many systems exhibit behaviors where the system’s possible state locations can be deduced to a set of states, or that a system can be derived to a situation where its possible system state locations can be deduced to a set of states.

Assumptions

- **FAULTS ARE UNOBSERVABLE** \((\Sigma_f \subseteq \Sigma_u \subseteq \Sigma)\)
  
  Otherwise their detection would be trivial

- **MODELLED FAULTS DO NOT BRING THE SYSTEM TO A HALT**
  
  Providing enough time to diagnose the fault before the system crashes

- **NO ARBITRARILY LONG STRINGS OF UNOBSERVABLE EVENTS**
  
  \((\forall suv \in \mathcal{L}_G, s, v \in \Sigma_0^*, u \in \Sigma_u^*, \exists n \in \mathbb{N}, \text{ such that } ||u|| \leq n)\)
  
  This ensures that following the occurrence of an unobservable event, sooner or later the system will produce an observable event. Otherwise the system may get stuck in an infinitely long unobservable string of events, which prevents any diagnosis about what is going on inside the system

- **LIVE LANGUAGE** \((\forall x \in X, \exists e \in \Sigma \text{ such that } \delta(x, e) \text{ is defined})\)
  
  This is to ensure that in the future the system will always produce a sufficiently long string of observable events to be used for diagnosis.
The developed algorithm produces diagnoser states $q \in Q_d$; each diagnoser state $q$ is a set of ordered pairs composed of the diagnoser’s estimates of system state $x$, and system condition $\ell$.

**System Condition Labels $L$**
- $\{N\}$: Normal Operation
- $F = \{F_1, F_2, \ldots, F_m\}$: Fault labels for fault type events $\Sigma_{f_i}$, $i = 1, \ldots, m$

**Diagnoser State Composition:**
$$q = \{(x_1, \ell_1), \ldots, (x_k, \ell_k)\} \quad x_j \in X, \quad \ell_j \in L = \{N\} \cup 2^F \quad j = 1, \ldots, k$$

**Functionality of Algorithm**
- **Offline**
  - Generate entire diagnoser
- **Online**
  - Provide on-line diagnostics

**Diagnoser state $q$ is considered an $F_i$-certain state if:**
$$F_i \in \ell_j \quad \forall j$$
The Developed Algorithm for Diagnoser Construction

Step 1: Constructing $q_0$

i. Since it is assumed that the initial state of the original system $G$ is normal (non-faulty), the algorithm begins by setting $q_s = \{(x_0, N)\}$.

ii. Next, $q_s$ is extended to include its unobservable reach set by $q_s = q_s \cup \{(x, \ell)|x \in \delta(x_0, u), u \in \Sigma^*, \ell = V(\{N\}, u)\}$ where $V$ is a labeling function.

iii. Next, $q_s$ is recursively extended to all of the plant’s reachable states.

iv. The diagnoser’s initial state $q_0 \subseteq q_s$ can be obtained as $q_0 = \{(x, \ell)|(x, \ell) \in q_s, x \in UR(X_s)\}$.

Step 2: Constructing remaining diagnoser states $q \in Q_d$

Starting with $q_0$, Step 2 constructs the remaining accessible states of the diagnoser $q \in Q_d$ by

$$
\delta_d(q, e) = \bigcup_{(x, \ell) \in q} \{(\delta(x, e), V(\ell, t))\}
$$

Algorithmic Construction: Example

Starting with the Original System, we will now construct its Semi-Asynchronous Diagnoser

Original System

Algorithm

Diagnoser?

\[ X_s = \{2, 9\} \]
\[ \Sigma_o = \{\alpha, \beta, \delta\} \]
\[ \Sigma_u = \{f\} \]
Step 1: Constructing $q_0$

I. $q_s = \{(x_0, N)\}$

II. $q_s = q_s \cup \{(x, \ell) | x \in \delta(x_0, u), u \in \Sigma^*_u, \ell = \nabla(\{N\}, u)\}$

III. $q_s$ is recursively extended to all of the plant’s reachable states

IV. $q_0 = \{(x, \ell) | (x, \ell) \in q_s, x \in UR(X_s)\}$

Original System

- $X_s = \{2, 9\}$
- $\Sigma_o = \{\alpha, \beta, \delta\}$
- $\Sigma_u = \{f\}$

I. $q_s = \{(1, N)\}$

II. $q_s = \{(1, N), (5, F)\}$

III. $q_s = \{(1, N), (5, F), (2, N), (3, N), (4, F), (5, F), (6, F), (7, N), (8, F), (9, F), (10, F)\}$

IV. $q_0 = \{(2, N), (9, F)\}$
Step 2: Constructing Diagnoser States $q \in Q_d$

Starting with $q_0$, construct the remaining accessible states of the diagnoser $q \in Q_d$ by

$$\delta_d(q, e) = \bigcup_{(x, \ell) \in q} \{(\delta(x, e), \nabla(\ell, t))\}$$

- $q_0 = \{(2, N), (9, F)\}$
- $q_1 = \{(3, N), (10, F), (7, N), (6, F)\}$
- $q_2 = \{(1, N), (8, F), (5, F)\}$
- $q_3 = \{(4, F)\}$

Original System

$X_s = \{2, 9\}$
$\Sigma_o = \{\alpha, \beta, \delta\}$
$\Sigma_u = \{f\}$
Algorithmically Constructed Diagnoser $G_d$

$G_d = (Q_d, \Sigma_d, \delta_d, q_0)$ a four-tuple representation of diagnoser generated by the algorithm

$Q_d \subseteq 2^{X \times L}$  
State space

$\Sigma_d = \Sigma_o$  
Event Set

$\delta_d: X \times \Sigma_d \rightarrow 2^X$  
State transition relation

$q_0$  
Initial State

$x = \{2, 9\}$  
$\Sigma_o = \{\alpha, \beta, \delta\}$

$\Sigma_u = \{f\}$

Original System

Diagnoser

$q_0 = \{(2, N), (9, F)\}$  
$q_3 = \{(4, F)\}$

$q_1 = \{(3, N), (10, F), (7, N), (6, F)\}$  
$q_2 = \{(1, N), (8, F), (5, F)\}$
The original system is in State 2, however, only having access to $X_s$, the diagnoser initially estimates the system to be in States 2 or 9.
Simulation

System Trace

Original System

Natural Projection

Observed System Trace

Semi-asynchronous Fault Diagnosis of Discrete Event Systems

Alejandro White
Simulation

System Trace

\( X_s = \{2, 9\} \)
\( \Sigma_o = \{\alpha, \beta, \delta\} \)
\( \Sigma_u = \{f\} \)

\( q_0 = \{(2, N), (9, F)\} \)
\( q_1 = \{(3, N), (10, F), (7, N), (6, F)\} \)
\( q_2 = \{(1, N), (8, F), (5, F)\} \)
\( q_3 = \{(4, F)\} \)

\( \beta \rightarrow \delta \rightarrow \alpha \)
\( 2 \rightarrow 3 \rightarrow 6 \)

Natural Projection
\( P: \Sigma^* \to \Sigma_o^* \)

Diagnoser does not update estimate as \( f \) is unobservable event

Observed System Trace

Semi-asynchronous Fault Diagnosis of Discrete Event Systems
Simulation

System Trace

\[ X_s = \{2, 9\} \]
\[ \Sigma_o = \{\alpha, \beta, \delta\} \]
\[ \Sigma_u = \{f\} \]

\[ q_0 = \{(2, N), (9, F)\} \]
\[ q_3 = \{(4, F)\} \]
\[ q_2 = \{(1, N), (8, F), (5, F)\} \]

Natural Projection
\[ P: \Sigma^* \rightarrow \Sigma_o^* \]

\[ \beta \quad f \quad \delta \]
\[ 2 \rightarrow 3 \rightarrow 6 \rightarrow 8 \]

Observed System Trace
Simulation

System Trace

\[ X_s = \{2, 9\} \]
\[ \Sigma_o = \{\alpha, \beta, \delta\} \]
\[ \Sigma_u = \{f\} \]

Observed System Trace

\[ q_0 = \{(2, N), (9, F)\} \]
\[ q_1 = \{(3, N), (10, F), (7, N), (6, F)\} \]
\[ q_3 = \{(4, F)\} \]

Natural Projection

\[ P: \Sigma^* \rightarrow \Sigma_o^* \]
\[ q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_0 \]

Alejandro White

Semi-asynchronous Fault Diagnosis of Discrete Event Systems
Simulation

System Trace

Diagnoser reaches and remains at $q_3$, an $F_i$-certain state

Observed System Trace

Natural Projection $P: \Sigma^* \rightarrow \Sigma_o^*$

Alejandro White
Semi-asynchronous Fault Diagnosis of Discrete Event Systems

$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_0 \rightarrow q_3$

$q_1 = \{(3, N), (10, F), (7, N), (6, F)\}$

$q_2 = \{(1, N), (8, F), (5, F)\}$

$q_3 = \{(4, F)\}$
Are All System Faults Diagnosable?

The proposed algorithm will create a Semi-Asynchronous Diagnoser $G_d$ for any provided system $G$

Will the created Semi-Asynchronous Diagnoser $G_d$ diagnose all modelled system fault occurrences
Fault Occurrence Does Not Lead To $F_i$ — Certain State

System Trace:
$q_2, q_3, q_4$, never reaching $f_2$ certain state

Observed System Trace:
$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_2 \rightarrow q_3 \ldots$

Diagonser cycles arbitrarily in states

Alejandro White
Semi-asynchronous Fault Diagnosis of Discrete Event Systems
Definition: The plant $G$ with the live language $\mathcal{L}_G$, is said to be $F_i$-semi-asynchronously diagnosable with respect to the failure type $\Sigma_{f_i}$, the natural projection $P$, and the initial set of estimated system states $X_S$, if and only if

for all $s_2 \in \text{Post}(X_S)$ and $f \in \Sigma_{f_i}$ with $f \in s_2$, there exists an upper bound $n_i \in \mathbb{N}$ such that for any $s_1, s_3$ with $s_1.s_2 \in \mathcal{L}_G, s_3 \in \mathcal{L}_{G/s_1.s_2}, \|s_3\| \geq n_i$, the following condition holds:

$\{u, v \in \Sigma^* \text{ with } u, v \in \mathcal{L}_G, u \in \text{Pre}(X_S) \text{ and } v \in P^{-1}_{\text{ext}(L_G)}(P(s_2.s_3)) \rightarrow f \in uv\}$

CASE 1

for all $s_1 \in \text{Pre}(X_S)$ and $f \in \Sigma_{f_i}$ with $f \in s_1$, there exists an upper bound $n_i \in \mathbb{N}$ such that for any $s_2 \in \mathcal{L}_G, s_2 \in \mathcal{L}_{G/s_1}$, with $\|s_2\| \geq n_i$, the following condition holds:

$\{u, v \in \Sigma^* \text{ with } u, v \in \mathcal{L}_G, u \in \text{Pre}(X_S) \text{ and } v \in P^{-1}_{\text{ext}(L_G)}(P(s_2)) \rightarrow f \in uv\}$

CASE 2
Example: Semi-Asynchronous Diagnosability

Let $X_S = \{2,9\}$

Upon reaching state $x = 2$, the system successively follows the sequence

$u = \alpha, \quad v = \alpha\beta(\alpha\delta\beta)^* \quad u.v \in \mathcal{L}_G$

$u \in \text{Pre}(X_S), \quad v \in \text{Pre}_{\text{ext}}(\mathcal{L}_G)(P(s_2.s_3))$

$\forall \quad f_2 \notin uv \Rightarrow F_t$-Semi-Asynchronous Diagnosability is VIOLATED for CASE 1

$\Rightarrow G_2$ is not $F_2$-Semi-Asynchronously Diagnosable with respect to $X_S$
Conclusion

- In this presentation, we have introduced a novel diagnoser that may be activated asynchronously while the system under diagnosis is in operation.
- Provided an algorithm for constructing the diagnoser
- Introduced the concept of Semi-Asynchronous Diagnosability, in a formal definition
- Provided the conditions for Semi-Asynchronous Diagnosability with respect to diagnoser activation
- Relative to a set of estimated system state locations $X_s$; the constructed diagnoser is capable of diagnosing fault occurrences in the system without access to information on system operation prior to the system reaching $X_s$
Future Work

- Develop systematic methods to verify the semi-asynchronous diagnosability of a given system
- Extension to distributed and/or decentralized architectures for scalability
Acknowledgements

- My teammates at ACCESS Laboratory and TECHLAV Center

- Financial support from Title III

- Support from Air Force Research Laboratory and Office of the Secretary of Defense under agreement number FA8750-15-2-0116

- Support from US ARMY Research Office under agreement number W911NF-16-1-0489
Questions