Augmenting the observers: diagnosability of prefix-closed languages, diagnosers and the fault detection for finite state automata

From observability to diagnosis in discrete event systems

Prof. Francesco BASILE
Email: fbasile@unisa.it

December 2020

UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II
DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELLE TECNOLOGIE DELL'INFORMAZIONE

Course syllabus

1. Discrete Event Systems (DES), Languages and Automata
2. Petri nets (PNs) and their twofold representation to model DES
3. MILP and ILP formulations: logical conditions, binary variables “do everything", and variable connecting
4. Adding uncertainty: unobservable events and observers for finite state automata and PNs
5. **Augmenting the observers: diagnosability of prefix-closed languages, diagnosers and the fault detection for finite state automata**
6. Diagnosability and fault detection in PNs - Part I: graph-based approaches
7. Diagnosability and fault detection in PNs - Part II: algebraic approaches for bounded systems
8. Security issues in DES: non-interference and opacity
9. Non-interference and opacity enforcement
10. Open issues

Outline

The compiled diagnoser

- To implement a fault detection algorithm an agent which gives, after each observed event, a set of faults that could have happened, or a set of fault states that the system could have reached, must be obtained. Such an agent is called *diagnoser*.
- A standard approach is to build a DES, called *compiled diagnoser*. At each state transition this system provides the set of faults that could have happened (SampathLafortune95), or a set of fault states that the system could have reached (ZadKwongWonham03).

The compiled diagnoser

- In general, the compiled diagnoser building is very computation demanding, even if it can be performed offline, and its state space results to be big. However, the computational effort to run a diagnoser is very low.
- The compiled diagnoser provides a fast on-line diagnosis at a price of excessive memory requirements since all the diagnoser state space must be available, and hence, it may be applied only when the state spate is bounded. This is the usual approach when the plant is modeled as finite state automata.

The interpreted diagnoser

- Another approach is to write an algorithm, called *interpreted diagnoser*, which online, after each observed event, computes the set of faults that could have happened, or a set of fault states that the system could have reached.
- In this case, the computational effort to run the diagnoser is bigger than in the case of the compiled diagnoser and it is difficult to derive a diagnosability test, while the memory requirement is much less since there is no need to precompute any state space. This approach is used in particular when the plant is modeled by Petri nets, since their mathematical representation mitigates the effort of the online computation.

The mixed approach

- A mixed approach is to write an algorithm, which online, after each observed event, computes the set of faults that could have happened, or a set of fault states that the system could have reached, on the basis of a system/graph computed offline.
- In this case, a trade-off between offline and online computation is pursued. However, memory requirements are most significant with respect to the interpreted diagnosed case.

The assumptions

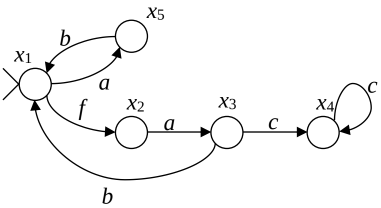
- The system to be diagnosed is modeled as a *finite state automaton G*.
- The model G includes both the normal and the faulty behavior. The set of events E is partitioned as $E = E_o \cup E_u$ in two disjoint subsets, where E_o is the set of observable events and E_u is the set of unobservable events.
- Let $E_f \subseteq E$ denote the set of failure events which are to be diagnosed. Let us assume that $E_f \subseteq E_u$, since it is straightforward diagnose an observable failure event.

Two main assumptions are usually done:

(A1) The language L generated by G is live. This means that there not exists a state $x \in X$ from which no event is possible.

(A2) There does not exist in G any cycle of unobservable events.

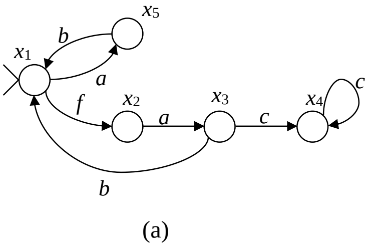
Augmenting the observer

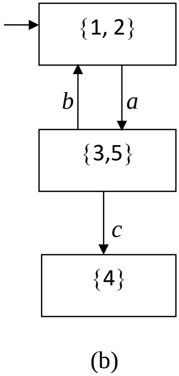


G

The set of observable events is $E_o = \{a, b, c\}$ and the set of unobservable events, that is equal to the set of fault events, is $E_u = E_f = \{f\}$.

Augmenting the observer

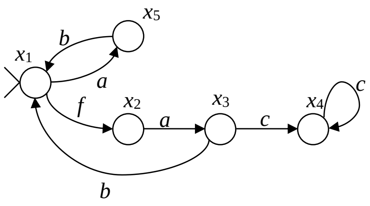




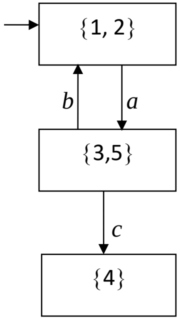
(a) G and (b) its observer

The initial state of the observer is $x_{0,obs} = \{1, 2\}$. It means that if no event is observed the system G can be either in state 1 or in state 2...

Augmenting the observer



(a)



(b)

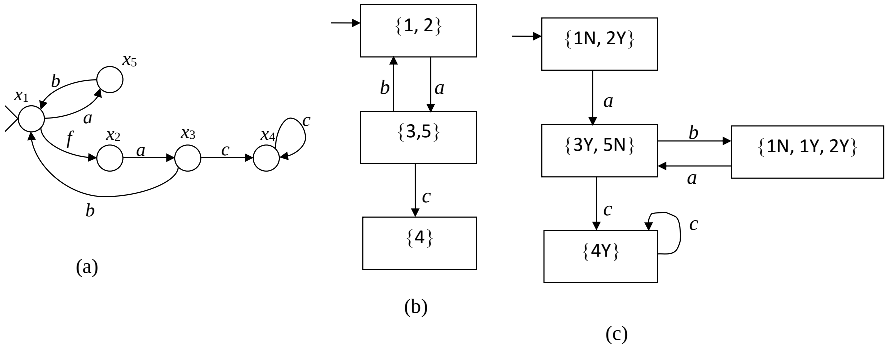
(a) G and (b) its observer

After observing string $t = a$, we do not know if the system has the fault f is occurred or not, but, after observing $w = ac$, we know with certainty that f must have occurred.

Augmenting the observer

- Diagnosis problem is the problem of associate to each observed string of events a diagnosis state, such as “normal” or “faulty” or “uncertain”.
- The uncertainty can be reduced continuing to make observations.
- We can automate this kind of inferencing about the past constructing an automaton that is similar to the observer, but that contains additional information regarding the occurrence of fault transitions. We call this modified observer a *diagnoser automaton*.

Augmenting the observer



(a) an automaton G, (b) its observer and (c) its diagnoser

Label N means that "f has not occurred yet" while Y means that "yes, f has occurred".

An algorithm to compute the diagnoser

The key modifications to the construction of $Obs(G)$ for the purpose of building $Diag(G)$:

M1. When building the unobservable reach of the initial state x_0 of G:

(a) Attach the label N to states that can be reached from x_0 by unobservable strings in $[E_u \setminus f]^*$;

(b) Attach the label Y to states that can be reached from x_0 by unobservable strings that contain at least one occurrence of f;

(c) If state z can be reached both with and without executing f, then create two entries in the initial state set of $Diag(G)$: zN and zY.

An algorithm to compute the diagnoser

M2. When building subsequent reachable states of $Diag(G)$:

(a) Follow the rules for the transition function of $Obs(G)$, but with the above modified way to build unobservable reaches with state labels;

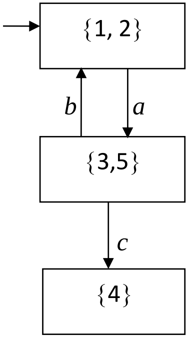
(b) Propagate the label Y. Namely, any state reachable from state zY should get the label Y to indicate that f has occurred in the process of reaching z and thus in the process of reaching the new state.

M3. No set of marked states is defined for $Diag(G)$.

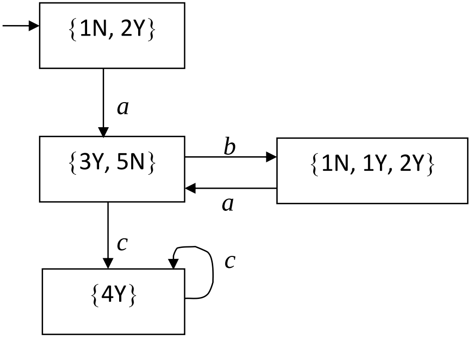
Diagnoser

- Diagnoser $Diag(G)$ is a deterministic automaton, whose set of events is $E = E_o$ and that generates a language $\mathcal{L}(Diag(G)) = P[L(G)]$.
- Each state of $Diag(G)$ is a subset of $X \times \{N, Y\}$.
- Modification $M1(c)$ implies that if a state can be reached by two paths having the same observable projection and such that one path contains the fault f and the other one does not in a node of $Diag(G)$ will exist two pairs xN and xF. It means that the cardinality of $Diag(G)$ is always greater than or equal to the cardinality of $Obs(G)$.

Diagnoser and observer



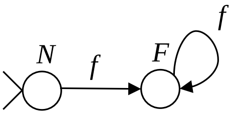
(a)



(b)

An alternative way to build the diagnoser

An alternative way to build the diagnoser is to consider the following label automation A_{lab}



and to compute $G \parallel A_{lab}$. The diagnoser can be obtained as $Obs(G \parallel A_{lab})$. Indeed, it is equivalent to the observer of $G \parallel A_{lab}$.

Fault detection

We can perform diagnosis by examination of the diagnoser states. In $Diag(G)$ we can distinguish three different kind of states:

- *negative state:* if in the node of $Diag(G)$ for all pairs (x, l) , $l = N$. Thus reaching this node we are sure that fault f has not occurred yet;
- *positive state:* if in the node of $Diag(G)$ for all pairs (x, l) , $l = Y$. Thus reaching this node we are sure that fault f has occurred;
- *uncertain state:* if in the node of $Diag(G)$ there exists at least one pair (x, l) such that $l = N$ and at least one pair (x, l), such that $l = Y$. Thus, we cannot say nothing about the occurrence of fault f.

Diagnosable fault

Consider a DES system G and event set $E = E_u \cup E_o$, where E_o denotes the observable events and E_u denotes the unobservable ones, and a fault event $f \subseteq E_u$.

Let L be the live and prefix-closed language generated by G.

L/σ is the post-language of L after the sequence of transitions σ, i.e. $L/\sigma = \left\{ v \in T^* \text{ s.t. } \sigma v \in L \right\}$.

A sequence $v \in L/\sigma$ is called *continuation* of σ.

Let $Pr : T^* \mapsto T_o^*$ be the usual projection, which erases the unobservable transitions in a sequence u. The inverse projection operator Pr_L^{-1} is defined as

$$Pr_L^{-1}(r) = \left\{ \sigma \in L \text{ s.t. } Pr(u) = r \right\}.$$

Diagnosable fault

Definition (Lafortune95)

A fault event f is said to be diagnosable if

$$\exists\ h \in \mathbb{N} \text{ such that } \forall\ \sigma = uf \textit{ with } f \notin u, \text{and } \forall\ v \in L/\sigma \textit{ with } |v| \geq h\,,$$

it is

$$r \in P_L^{-1}\big(P(\sigma v)\big) \Rightarrow f \in r\,.$$

A fault f is diagnosable if for every trace s ending with f, there exists a sufficiently long continuation v such that any other trace indistinguishable from σv (producing the same record of observable events) contains f (are also faulty).

Assumption (A1) allows us to state that when the property of diagnosability is satisfied, we are sure that if f occurs, then $Diag(G)$ will enter a positive state in a bounded number of events after the occurrence of f.

Checking diagnosability

Definition (Lafortune95)

Let us consider a system G and its diagnoser $Diag(G)$. We say that a cycle in $Diag(G)$ is an *indeterminate cycle* if it is composed exclusively of uncertain states for which there exist:

- a corresponding cycle (of observable events) in G involving only states that carry Y in their labels in the cycle in $Diag(G)$ and
- a corresponding cycle (of observable events) in G involving only states that carry N in their labels in the cycle in $Diag(G)$. ■

The notion of indeterminate cycles is very important because their analysis gives us necessary and sufficient conditions for diagnosability and gives a method to test the property.

Multiple fault sets

In the general case, the set of fault events is partitioned into *m* disjoint subsets that represent the set of fault classes:

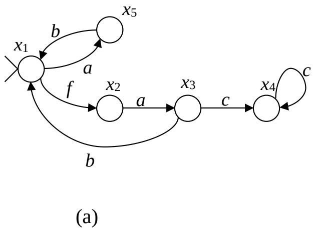
$$E_f = E_{f1}, E_{f2}, \ldots, E_{fm}.$$

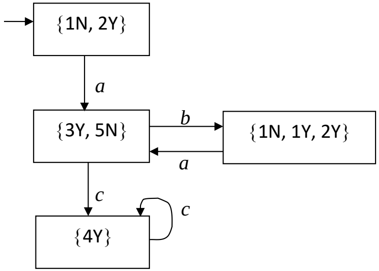
The aim is that of identifying the occurrence, if any, of failure events, given the set of generated words containing only observable events.

Proposition (Lafortune95)

A language $\mathcal{L}$ without multiple failures of the same type is diagnosable *if and only if its diagnoser $Diag(G)$ has no indeterminate cycles for all failure types E_{fi}.*

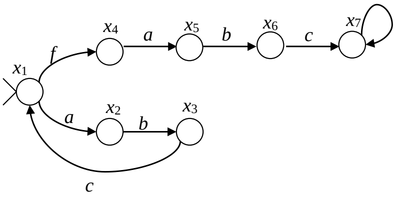
Indeterminate cycles

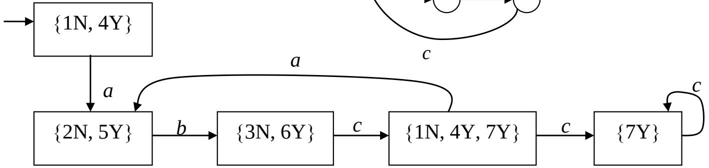




The diagnoser has a potential indeterminate cycle. Let us verify if conditions of Definition 2 are satisfied. In G there exists the cycle " $1 \to 2 \to 3 \to 1$" involving only states that carry Y in their labels and there exists another cycle " $1 \to 5 \to 1$" involving only states that carry N in their labels. Thus, this cycle is indeterminate thus the fault is not diagnosable.

Indeterminate cycles



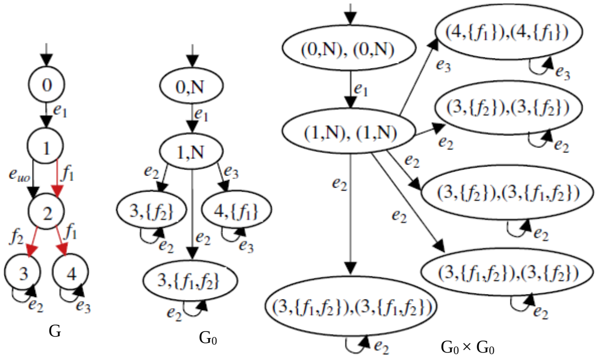


The diagnoser has a potential indeterminate cycle. However the only cycle that can cause the diagnoser to remain in its cycle of uncertain states is the cycle “ $1 \to 2 \to 3 \to 1$”, and these states all have the N label in the corresponding diagnoser states. The cycle of uncertain states in the diagnoser is therefore not indeterminate.

Twin machine technique

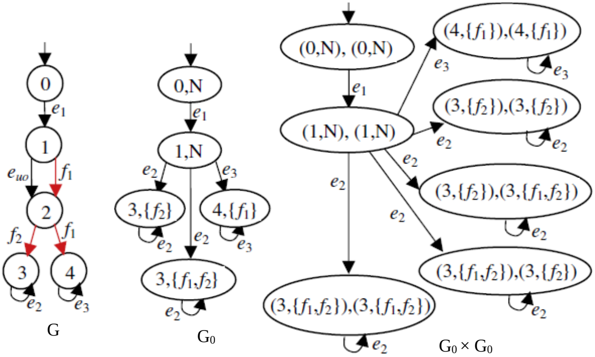
- The construction of the entire diagnoser may be very computation demanding.
- Its size is exponential in the number of states of G, as well as in the number of faults if a single diagnoser is desired. This second limitation can be addressed by building separate diagnosers for each fault type.
- The first limitation can be addressed by using the “twin machine” technique.
- The idea behind this technique is that a fault f is diagnosable if and only if there is no pair of arbitrarily long traces having the same observable projection, such that f occurs in the first trace but not in the second.

Twin machine technique



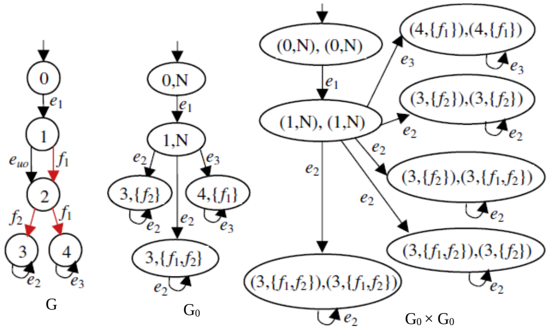
First, a nondeterministic observer of G, denoted by G_0, whose states can be reached by taking only the observable transitions is computed: a label f_i in its state indicates that fault f_i occurs along a certain path from the initial state to this state; otherwise, the state label is “N”.

Twin machine technique



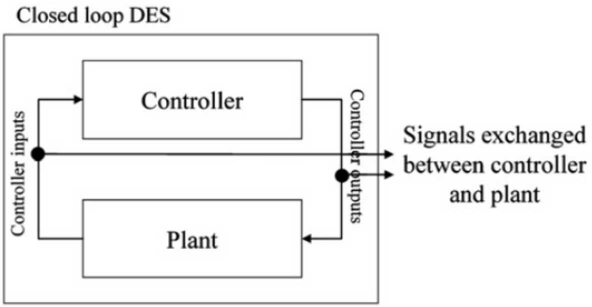
The next step is to compute the parallel composition of G_0 with itself. The system is not diagnosable if $G_0 \times G_0$ contains a cycle where the left labels and the right labels differ by fault type f_i in every state along the cycle (they may differ in other ways as well).

Twin machine technique

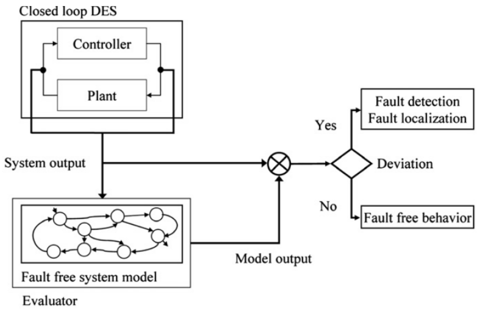


In $G_0 \times G_0$ there is a self-loop at the state $((3, f_2), (3, f_1, f_2))$ or at the state $((3, f_1, f_2), (3, f_2))$. In each case, the cycle indicates that fault f_1 is not diagnosable in G, as the presence of each cycle implies the existence of two arbitrarily long traces with the same projection, where f_1 is contained in one trace but not in the other.

The DES extended process

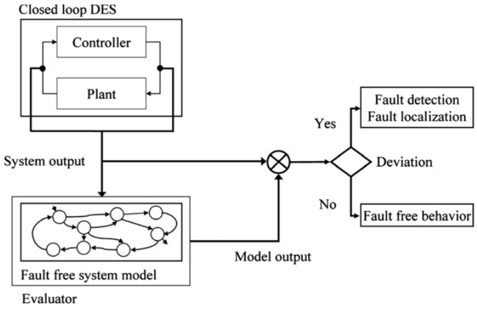


FDI principle for closed loop DES



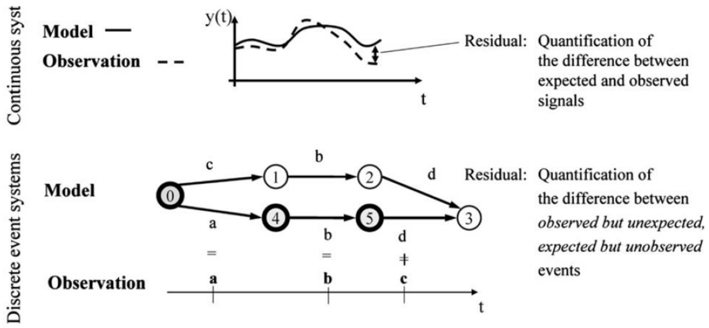
In the context of continuous time systems, diagnosis in consists in the determination of the time of detection, kind, size and location of a fault. Diagnosis in a wide sense includes fault detection, isolation and identification.

FDI principle for closed loop DES



In the context of discrete event systems, kind and size has not sense, and the diagnosis task may be considered to be completed after fault detection and localization.

The concept of residual



Continuous syst
Model
Observation
y(t)
t
Residual: Quantification of the difference between expected and observed signals
Discrete event systems
Model
Observation
c
b
d
a
b
d
=
=
≠
a
b
c
0
1
2
3
4
5
t
Residual: Quantification of the difference between *observed but unexpected*, *expected but unobserved* events

A residual is a fault indicator, based on a deviation between measurements and model-based computation.

The concept of residual

Denote by

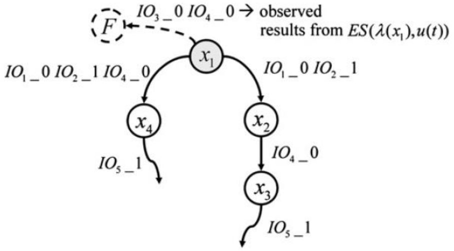
- $ES(u(i), u(k))$ the set of rising and falling edges between two I/O vector sets $u(i)$ and $u(k)$, called evolution set;
- $\lambda(\tilde{x})$ the I/O vector set generated by the fault free model of the extended process at state $\tilde{x}$;
- $f(\tilde{x})$ the transition function of the fault free model of the extended process at state $\tilde{x}$.

An example of residual is

$$Res1(\tilde{x}, u(t)) = ES(\lambda(\tilde{x}), u(t)) \setminus \bigcup_{\forall x' \in f(\tilde{x})} ES(\lambda(\tilde{x}), \lambda(x'))$$

$\bigcup_{\forall x' \in f(\tilde{x})} ES(\lambda(\tilde{x}), \lambda(x'))$ is the union of the sets of rising and falling edges when the last estimated current state and each of its direct successor states are considered, it is the overall expected behavior of the system. Hence $Res1(\tilde{x}, u(t))$ represents an unexpected behavior of the system.

Unexpexted event



$IO_3\_0\ IO_4\_0 \rightarrow$ observed
results from $ES(\lambda(x_1),u(t))$
F
x_1
$IO_1\_0\ IO_2\_1\ IO_4\_0$
$IO_1\_0\ IO_2\_1$
x_4
x_2
IO_5_1
IO_4_0
x_3
IO_5_1

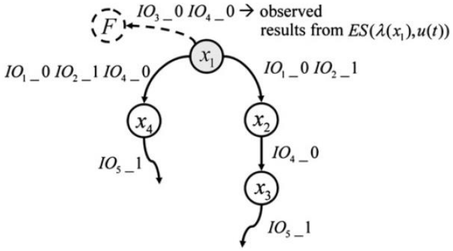
Let $\tilde{x} = x_1$,
$Res1(\tilde{x}, u(t)) =$
$\{IO_3\_0, IO_4\_0\} \setminus (\{IO_1\_0, IO_2\_1, IO_4\_0\} \bigcup \{IO_1\_0, IO_2\_1\}) = \{IO_3\_0\}$.
This result that implies that the system operator should check the sensor or actuator that is connected with IO_3.
If the fault is not located at the sensor/actuator connected with IO_3, one may consider a stricter formulation of the expected behavior.

Unexpexted event

$$Res2(\tilde{x}, u(t)) = ES(\lambda(\tilde{x}), u(t)) \setminus \bigcap_{\forall x' \in f(\tilde{x})} ES(\lambda(\tilde{x}), \lambda(x'))$$

The intersection delivers the I/O edges that must be observed no matter which following state in the model is taken, obviously $Res1(\tilde{x}, u(t)) \subseteq Res2(\tilde{x}, u(t))$.

Unexpected event



$IO_3\_0\ IO_4\_0 \rightarrow$ observed
results from $ES(\lambda(x_1),u(t))$
F
x_1
$IO_1\_0\ IO_2\_1\ IO_4\_0$
$IO_1\_0\ IO_2\_1$
x_4
x_2
IO_5_1
IO_4_0
x_3
IO_5_1

$Res2(\tilde{x}, u(t)) = \{IO_3\_0, IO_4\_0\} \setminus$
$(\{IO_1\_0, IO_2\_1, IO_4\_0\} \bigcap \{IO_1\_0, IO_2\_1\}) = \{IO_3\_0, IO_4\_0\}$.
This result that implies implies that the occurrence of a change in value of IO_4 is not always expected and thus it is another possible fault localization.

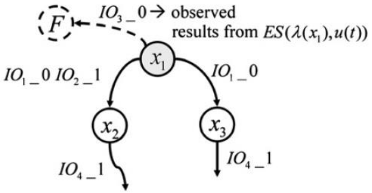
Missed event

$$Res3(\tilde{x}, u(t)) = \bigcap_{\forall x' \in f(\tilde{x})} ES(\lambda(\tilde{x}), \lambda(x')) \setminus ES(\lambda(\tilde{x}), u(t)).$$

*Res*3 is the set difference of the I/O edges that are expected no matter which following state is taken and the I/O edges that have been observed. It characterizes a missed behavior, a less restrictive version is

$$Res4(\tilde{x}, u(t)) = \bigcup_{\forall x' \in f(\tilde{x})} ES(\lambda(\tilde{x}), \lambda(x')) \setminus ES(\lambda(\tilde{x}), u(t)).$$

Missed event



F
$IO_3\_0 \rightarrow$ observed results from $ES(\lambda(x_1),u(t))$
x_1
$IO_1\_0\ IO_2\_1$
IO_1_0
x_2
x_3
IO_4_1
IO_4_1

Let $\tilde{x} = x_1$,
$Res3(\tilde{x}, u(t)) = \{IO_1\_0\}$, $Res4(\tilde{x}, u(t)) = \{IO_1\_0, IO_2\_1\}$.

First, the component connected with IO_1 should be checked. If a fault is not found at this component, the component connected with IO_2 must be checked.

References

- Diagnoser and Diagnosability – Chapter 2 (section 2.5.3) in
  - C. G. Cassandras and S. Lafortune
    Introduction to Discrete Event Systems
    Springer, 2008
- Diagnoser and Diagnosability – the seminal works
  - M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, D. Teneketzis
    Diagnosability of Discrete-Event Systems
    IEEE Transaction on Automatic Control, 1995
  - S. H. Zad, R. H. Kwong, and W. M. Wonham
    Fault diagnosis in discrete- event systems: Framework and model reduction
    IEEE Transaction on Automatic Control, 2003

References

- Fault diagnosis (faults built in the model)

M. Sampath, R. Sengupta, S. Lafortune, and K. Sinnamohideen
Failure diagnosis using discrete event models
IEEE Trans. Control System Technology, 1996

- Fault diagnosis (fault free model)

M. Roth, J.J. Lesage, L. Litz
The concept of residuals for fault localization in discrete event systems
Control Engineering Practice, 2011

Augmenting the observers: diagnosability of prefix-closed languages, diagnosers and the fault detection for finite state automata

From observability to diagnosis in discrete event systems

Prof. Francesco BASILE
Email: fbasile@unisa.it

December 2020

UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II
DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELLE TECNOLOGIE DELL'INFORMAZIONE