UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria dell'Informazione

Diagnostica degli eventi

Implementazione in Matlab

Events diagnosis

Implementation in Matlab

Candidato:
Giuseppe Di Mauro

Relatore: Chiar.mo/a
Prof. Silvia Maria Zanoli

Anno Accademico 2021-2022

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea in Ingegneria dell'Informazione
Via Brecce Bianche – 60131 Ancona (AN), Italy

Sommario

Questa tesi ha la scopo di illustrare i concetti fondamentali della diagnostica di sistemi ad eventi discreti (DES) e di sviluppare un software per l'applicazione delle metodologie della fault diagnosis (diagnosi di un guasto).

Il problema della diagnosi consiste nell'individuare se e quando si verifica un comportamento anomalo nel sistema considerato. L'approccio presentato in questa tesi utilizza come modello gli automi ipotizzando che alcune degli eventi siano non osservabili, in particolare quelli che modellano i guasti.

La definizione standard di diagnosability dei sistemi ad eventi discreti affronta il problema del rilevamento di fault non osservabili, utilizzando un modello costruito a partire da sequenze di eventi osservabili.

Un ulteriore concetto è quello della safe diagnosability, che richiede che il rilevamento dei fault avvenga prima dell'esecuzione di un determinato insieme di stringhe vietate durante il funzionamento del sistema. Ad esempio, questo vincolo potrebbe essere necessario per evitare che fault locali si trasformino in guasti che potrebbero causare rischi per la sicurezza.

Se il sistema è safe diagnosable, al rilevamento di un fault potrebbero essere forzate azioni di riconfigurazione, prima dell'esecuzione di comportamenti non sicuri.

Indice

Elenco delle figure

Premessa

Nell'elaborazione di questa tesi, ho deciso di non effettuare la traduzione di alcuni termini consolidati nella letteratura scientifica inglese.

Inoltre, nel paragrafo seguente, riporto tutti gli acronimi utilizzati nella relazione.

Acronimi

DES discrete event system;

ED event driven;

FSA finite-state automaton;

DFA deterministic finite-state automaton;

NFA non-deterministic finite-state automaton.

Introduzione

Questa tesi ha la scopo di illustrare i concetti fondamentali della diagnostica di sistemi ad eventi discreti (DES) e di sviluppare un software per l'applicazione delle metodologie della fault diagnosis (diagnosi di un guasto).

I DES (Sistemi ad eventi discreti) sono ED (Event Driven), ovvero sistemi la cui dinamica è guidata dall'occorrenza degli eventi stessi.

Nel nostro caso di studio, ci occuperemo di DES non temporizzati e a stati finiti. Il loro comportamento è modellabile attraverso un linguaggio regolare.

Il problema della diagnosi consiste nell'individuare se e quando si verifica un comportamento anomalo nel sistema considerato. L'approccio presentato in questa tesi utilizza come modello gli automi ipotizzando che alcune degli eventi siano non osservabili, in particolare quelli che modellano i guasti.

La definizione standard di diagnosability dei sistemi ad eventi discreti affronta ilproblema del rilevamento di fault non osservabili, utilizzando un modello costruito a partire da sequenze di eventi osservabili.

Un ulteriore aspetto è quello della safe diagnosability, che richiede che il rilevamento dei fault avvenga prima dell'esecuzione di un determinato insieme di stringhe vietate durante il funzionamento del sistema. Ad esempio, questo vincolo potrebbe essere necessario per evitare che fault locali si trasformino in guasti che potrebbero causare rischi per la sicurezza.

Se il sistema è safe diagnosable, al rilevamento di un fault potrebbero essere forzate azioni di riconfigurazione, prima dell'esecuzione di comportamenti non sicuri.

In questa tesi, verranno esposti tutti i concetti necessari a comprendere il lavoro svolto e verranno approfonditi i concetti fondamentali della diagnostica di DES (diagnosability e safe diagnosability) e verrà illustrato il software che ho sviluppato per l'applicazione delle metodologie della fault diagnosis.

Il presente lavoro di tesi è organizzato nel seguente modo.

Nel capitolo 1 vengono introdotti i concetti di automa e delle principali tecniche di composizione; inoltre vengono chiariti i concetti di automa deterministico e non deterministico.

Nel capitolo 2 vengono introdotti i concetti di osservatore e viene illustrata la procedura per la costruzione di un osservatore di un automa non deterministico e, dopo aver introdotto il concetto di DES parzialmente osservabile, la procedura per la costruzione di un osservatore di un automa con venti non osservabili.

Successivamente ci soffermeremo sul diagnoser (capitolo 3), usato per risolvere problemi di event diagnosis. I diagnoser sono una variazione degli osservatori e

vengono utilizzati per tenere traccia del comportamento del sistema e diagnosticare, se possibile, l'occorrenza degli eventi non osservabili di interesse. Inoltre, vengono approfonditi gli argomenti cardine della tesi: la diagnosability e la safe diagnosability.

Nel capitolo 4 viene fatta una panoramica sui software utilizzati nell'esperimento, tra cui il software da me sviluppato per l'applicazione delle metodologie della fault diagnosis.

Nel capitolo 5 analizzeremo due casi: il primo è un esercizio teorico per verificare il corretto funzionamento degli algoritmi implementati, il secondo è un esempio applicativo nel campo dell'automatica in cui vengono evidenziate l'utilità e la potenzialità di questo progetto.

Infine, nel capitolo 6, vengono proposte le conclusioni ed alcuni suggerimenti per eventuali progetti futuri, che si appoggino a questo.

Capitolo 1

Automa

L'automa è un modello matematico di calcolo che permette di descrivere con precisione e in maniera formale il comportamento di molti sistemi. Grazie alla sua semplicità e chiarezza questo modello è molto diffuso nell'ingegneria e nelle scienze, soprattutto nel campo dell'automatica, dell'informatica e della ricerca operativa.

Gli automi vengono utilizzati per descrivere linguaggi formali e per questo sono chiamati accettori o riconoscitori di un linguaggio.

L'insieme dei possibili simboli che possono essere forniti ad un automa costituisce il suo alfabeto.

Una sequenza di simboli (detta anche stringa o parola) appartiene al linguaggio se essa viene accettata dal corrispondente automa, ovvero se porta l'automa in uno stato valido, che sia lo stesso o un altro stato. Un sottoinsieme del linguaggio riconosciuto, chiamato linguaggio marcato porta l'automa dal suo stato iniziale ad uno stato finale o marcato.

A diverse classi di automi corrispondono diverse classi di linguaggi, caratterizzate da vari livelli di complessità.

L'usuale rappresentazione grafica di un automa a stati finiti è il grafo orientato.

1.1 Automa deterministico

Un automa a stati finiti deterministico (o deterministic finite-state automaton DFA) è un automa a stati finiti dove per ogni coppia di stati esiste al più una transizione.

Un automa deterministico, denotato con G, è una sestupla

$$G = (X, E, f, \Gamma, x_0, X_m)$$

dove:

- X è l'insieme degli stati;
- E è l'insieme finito degli eventi associati a G;
- $f : X \times E \to X$ è la funzione di transizione. $f(x, e) = y$ significa che c'è una transizione etichettata dall'evento e dallo stato x allo stato y; in genere, f è una funzione parziale nel suo dominio;

- $\Gamma : X \to 2^E$ è la funzione degli eventi attivi. $\Gamma(x)$ è l'insieme di tutti gli eventi e per cui $f(x, e)$ è definita

- x_0 è lo stato iniziale;

- $X_m \subseteq X$ è l'insieme degli stati marcati.

Formalmente, l'inclusione di Γ nella definizione è ridondante, poiché derivata da f.

Però nell'implementazione degli algoritmi è stata fondamentale.

Infatti la ricerca degli eventi attivi in un determinato stato x, ha permesso più volte di ridurre la complessità computazionale dell'algoritmo, evitando due casistiche principali: l'analisi ripetuta di uno stesso evento per un determinato stato, e l'analisi superflua di stati non raggiungibili.

1.2 Automa non deterministico

Nella definizione di automa deterministico, lo stato iniziale è uno stato singolo, tutte le transizioni hanno una label $e \in E$ e la funzione di transizione è deterministica, ovvero se un evento $e \in \Gamma(x)$, quindi è attivo, allora causa una transizione dallo stato x ad un unico stato $y = f(x, e)$.

Per scopi di modellazione e di analisi, è necessario rilassare questi requisiti.

In primo luogo, un evento e potrebbe portare a più stati diversi. O potrebbero verificarsi eventi che non sono osservabili e quindi raggiungere stati non previsti, definiamo questi eventi come ε-transitions. Nel secondo caso, quindi, è possibile che lo stato iniziale non sia più singolo, ma un insieme di stati. I motivi di ciò potrebbero essere la nostra ignoranza o il malfunzionamento di un componente del nostro sistema, o ancora l'assenza di un sensore che rilevi l'evento.

Abbiamo bisogno quindi di generalizzare la nozione di automa e definire l'automa non deterministico.

Un automa non deterministico (o non-deterministic finite-state automaton NFA), denotato con G_{nd}, è una sestupla

$$G_{nd} = (X, E \cup \{\varepsilon\}, f_{nd}, \Gamma, x_0, X_m)$$

dove i parametri hanno la stessa interpretazione nella definizione di automa deterministico, ad eccezione per:

- $f_{nd} : X \times E \cup \{\varepsilon\} \to 2^X$ è la funzione non deterministica che include l'evento ε, ovvero la transizione non osservabile. Il codominio è l'insieme potenza dello spazio di stato dell'automa;

- $x_0 \subseteq X$ potrebbe essere un insieme di stati.

1.3 Operazioni sugli automi

Esistono operazioni che si possono effettuare su un singolo automa (operazioni unarie) o su più automi (operazioni di composizione). Tra le prime possiamo citare: l'accessibilità, la coaccessibilità, il trim, il complemento, la proiezione e la proiezione inversa. Tra le composizioni di automi si trova il prodotto e la composizione in parallelo. Quest'ultima è particolarmente utile quando si vuole costruire il modello di un sistema molto complesso andando a combinare le sue singole parti.

1.3.1 Operazioni unarie

Accessibilità

Delle operazioni unarie è necessario introdurre il concetto di accessibilità.

Osserviamo che è possibile eliminare dall'automa G tutti gli stati che non sono raggiungibili dallo stato iniziale x_0, senza avere ripercussioni sul linguaggio generato e marcato dall'automa G.

Quando si elimina uno stato, occorre eliminare anche tutte le transizioni correlate a quello stato.

Questa operazione si definisce parte accessibile di G e si denota con $Ac(G)$.

Proiezione e proiezione inversa

Sia E l'insieme degli eventi di G. Consideriamo $E_s \subset E$.

Le proiezioni di $\mathcal{L}(G)$ e $\mathcal{L}_m(G)$ da E^* in E_s^*, $P_s[\mathcal{L}(G)]$ e $P_s[\mathcal{L}_m(G)]$, possono essere implementate in G sostituendo tutte le label delle transizioni in $E \setminus E_s$ con ε.

Il risultato è un automa non deterministico che genera e marca i linguaggi desiderati.

Riguardo la proiezione inversa, si consideri il linguaggio $K_s = \mathcal{L}(G) \subseteq E_s^*$ e $K_{m,s} = \mathcal{L}_m(G)$ e sia E_l un insieme di eventi più grande t.c. $E_l \supset E_s$.

Sia P_s la proiezione da E_l^* in E_s^*.

Un automa che genera $P_s^{-1}(K_s)$ e marca $P_s^{-1}(K_{m,s})$ può essere ottenuto aggiungendo auto-anelli a tutti gli stati di G per ogni evento in $E_l \setminus E_s$.

1.3.2 Operazioni di composizione

Prodotto

Consideriamo i due automi

$$G_1 = (X_1, E_1, f_1, \Gamma_1, x_{01}, X_{m1}) \; e \; G_2 = (X_2, E_2, f_2, \Gamma_2, x_{02}, X_{m2})$$

e siano questi accessibili.

Il prodotto di G_1 e G_2 è l'automa

$$G_1 \times G_2 := Ac(X_1 \times X_2, E_1 \cap E_2, f, \Gamma_{1\times 2}, (x_{01}, x_{02}), X_{m1} \times X_{m2})$$

dove

$$f((x_1, x_2), e) := \begin{cases} (f_1(x_1, e), f_2(x_2, e)) \text{ se } e \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ \text{non definita altrimenti} \end{cases}$$

quindi $\Gamma_{1\times 2}(x_1, x_2) = \Gamma_1(x_1) \cap \Gamma_2(x_2)$.

Si osservi che siamo interessati solo alla parte accessibile dell'automa.

Nel prodotto, le transizioni dei due automi devono essere sempre sincronizzate su un evento comune, quindi in $E_1 \cap E_2$. Inoltre gli stati sono coppie, in cui la prima componente è lo stato attuale di G_1 e la seconda è lo stato attuale di G_2.

È facilmente verificabile che:

$$\mathcal{L}(G_1 \times G_2) = \mathcal{L}(G_1) \cap \mathcal{L}(G_2)$$
$$\mathcal{L}_m(G_1 \times G2) = \mathcal{L}_m(G_1) \cap \mathcal{L}_m(G_2).$$

Il prodotto gode inoltre delle proprietà commutativa e associativa.

Anche nell'implementazione di questa operazione è stata fondamentale la funzione degli eventi attivi $\Gamma_{1\times 2}$, restringendo la ricerca degli stati raggiungibili: quest'ultima non ha analizzato tutti gli eventi in comune ai due automi, ma solo quelli che fossero anche attivi negli stati attuali di G_1 e G_2.

Composizione parallela

Consideriamo i due automi

$$G_1 = (X_1, E_1, f_1, \Gamma_1, x_{01}, X_{m1}) \; e \; G_2 = (X_2, E_2, f_2, \Gamma_2, x_{02}, X_{m2})$$

e siano questi accessibili.

La composizione parallela di G_1 e G_2 è l'automa

$$G_1 \| G_2 := Ac(X_1 \times X_2, E_1 \cup E_2, f, \Gamma_{1\|2}, (x_{01}, x_{02}), X_{m1} \times X_{m2})$$

dove

$$f((x_1, x_2), e) := \begin{cases} (f_1(x_1, e), f_2(x_2, e)) \text{ se } e \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ (f_1(x_1, e), x_2)) \text{ se } e \in \Gamma_1(x_1) \setminus E_2 \\ (x_1, f_2(x_2, e)) \text{ se } e \in \Gamma_2(x_2) \setminus E_1 \\ \text{non definita altrimenti} \end{cases}$$

quindi $\Gamma_{1\|2}(x_1, x_2) = [\Gamma_1(x_1) \cap \Gamma_2(x_2)] \cup [\Gamma_1(x_1) \setminus E_2] \cup [\Gamma_2(x_2) \setminus E_1]$.

I due automi sono anche qui sincronizzati sugli eventi comuni, che appartengono a $E_1 \cap E_2$.

Però, in questo caso, gli eventi privati, ovvero quelli in $(E_2 \setminus E_1) \cup (E_1 \setminus E_2)$, possono comunque essere eseguiti quando possibile. Questo tipo di composizione non è quindi restrittivo come il primo e per questo è più comunemente utilizzato.

Si osservi che se l'insieme degli eventi dei due automi coincide (quindi $E_1 = E_2$), allora la composizione parallela coincide con l'operazione di prodotto.

Per definire i linguaggi generati e marcati da $G_1 \| G_2$, consideriamo le due proiezioni

$$P_i : (E_1 \cup E_2) \to E_i \text{ per } i = 1, 2.$$

Allora otteniamo:

$$\mathcal{L}(G_1 \| G_2) = P_1^{-1}[\mathcal{L}(G_1)] \cap P_2^{-1}[\mathcal{L}(G_2)]$$
$$\mathcal{L}_m(G_1 \| G_2) = P_1^{-1}[\mathcal{L}_m(G_1)] \cap P_2^{-1}[\mathcal{L}_m(G_2)].$$

Anche la composizione parallela gode delle proprietà commutativa e associativa.

Capitolo 2

Osservatore

È sempre possibile trasformare un automa non deterministico in uno deterministico equivalente che generi e marchi lo stesso linguaggio. Questo automa prende il nome di osservatore e si denota con $Obs(G_{nd})$ o G_{obs}.

Gli stati di un osservatore sono la miglior stima degli stati dell'automa G_{nd}, da cui è stato costruito l'osservatore. La stima è basata sugli eventi osservati fino a quel momento.

Introduciamo il concetto di ε-reach dello stato x

$$\varepsilon R(x) := f_{nd}^{ext}(x, \varepsilon).$$

Non è altro che l'estensione della funzione di transizione dell'automa non deterministico f_{nd} nell'insieme di eventi E^* (chiusura di Kleene dell'insieme E).

Procedura per la costruzione di un osservatore $Obs(G_{nd})$ di un automa non deterministico G_{nd}

Sia $G_{nd} = (X, E \cup \{\varepsilon\}, f_{nd}, x_0, X_m)$ un automa non deterministico.
Allora $Obs(G_{nd}) = (X_{obs}, E, f_{obs}, x_{0,obs}, X_{m,obs})$ è costruito come segue:

1. Si definisce $x_{0,obs} := \varepsilon R(x_0)$. Si imposta $X_{obs} = \{x_{0,obs}\}$.
2. Per ogni $B \in X_{obs}$ ed $e \in E$, si definisce

   $f_{obs}(B, e) := \varepsilon R(\{x \in X : (\exists x_e \in B)[x \in f_{nd}(x_e, e)]\})$

   ogni qualvolta che $f_{nd}(x_e, e)$ è definita per qualche $x_e \in B$. In questo caso, si aggiunge lo stato $f_{obs}(B, e)$ a X_{obs}. Altrimenti se $f_{nd}(x_e, e)$ non è definita per ogni $x_e \in B$, allora $f_{obs}(B, e)$ non è definita.
3. Ripeti il passo 2 finché l'intera parte accessibile di $Obs(G_{nd})$ non è stata costruita.
4. $X_{m,obs} := \{B \in X_{obs} : B \cap X_m \neq \emptyset\}$.

Quindi, le proprietà di cui gode l'osservatore sono:

- $Obs(G_{nd})$ è un automa deterministico;

- $\mathcal{L}(Obs(G_{nd})) = \mathcal{L}(G_{nd})$;
- $\mathcal{L}_m(Obs(G_{nd})) = \mathcal{L}_m(G_{nd})$.

DES parzialmente osservabili

Abbiamo affrontato la nozione di ε-transitions per gli automi non deterministici, ovvero eventi non osservabili. Invece di etichettarli tutti con ε ed ottenere un automa non deterministico, definiamo questi eventi separatamente. Otterremo un automa deterministico il cui insieme degli eventi E sarà partizionato in due sottoinsiemi disgiunti: E_o, l'insieme degli eventi osservabili,ed E_{uo}, l'insieme degli eventi non osservabili. Quest'automa è noto come parzialmente osservabile.

A questo punto è possibile costruire l'osservatore di questo automa, il cui insieme degli eventi è $E = E_o \cup E_{uo}$.

È necessario definire però l'unobservable reach di ogni stato $x \in X$, denotato da $UR(x)$, come:

$$UR(x) = \{y \in X : (\exists t \in E_{uo}^*)[(f(x,t) = y)]\}.$$

Questa definizione viene poi estesa agli insiemi di stati $B \subseteq X$ come

$$UR(B) = \cup_{x \in B} UR(x).$$

Procedura per la costruzione di un osservatore $Obs(G)$ di un automa G con eventi non osservabili

Sia $G = (X, E, f, x_0, X_m)$ un automa deterministico e sia $E = E_o \cup E_{uo}$.
Allora $Obs(G) = (X_{obs}, E, f_{obs}, x_{0,obs}, X_{m,obs})$ è costruito come segue:

1. Si definisce $x_{0,obs} := UR(x_0)$. Si imposta $X_{obs} = \{x_{0,obs}\}$;
2. Per ogni $B \in X_{obs}$ ed $e \in E_o$, si definisce $f_{obs}(B, e) := UR(\{x \in X : (\exists x_e \in B)[x \in f(x_e, e)]\})$ ogni qualvolta che $f(x_e, e)$ è definita per qualche $x_e \in B$. In questo caso, si aggiunge lo stato $f_{obs}(B, e)$ a X_{obs}. Altrimenti se $f(x_e, e)$ non è definita per ogni $x_e \in B$, allora $f_{obs}(B, e)$ non è definita;
3. Ripeti il passo 2 finché l'intera parte accessibile di $Obs(G)$ non è stata costruita.
4. $X_{m,obs} := \{B \in X_{obs} : B \cap X_m \neq \emptyset\}$.

Per costruzione l'osservatore gode delle seguenti proprietà:

- $Obs(G)$ è un automa deterministico;
- $\mathcal{L}(Obs(G)) = P[\mathcal{L}(G)]$;
- $\mathcal{L}_m(Obs(G)) = P[\mathcal{L}_m(G)]$;

- Sia $B(t) = f_{obs}(x_{0,obs}, t)$. Allora $x \in B(t)$ se x è raggiungibile in G da una stringa in $P^{-1}(t) \cap \mathcal{L}(G)$.

Si osservi che per la costruzione di osservatori di automi non deterministici con eventi non osservabili, è sufficiente sostituire la funzione f con la funzione non deterministica estesa f_{nd}^{ext}.

Capitolo 3

Event diagnosis

In molte applicazioni, è interessante stabilire se gli eventi non osservabili potrebbero essersi verificati o devono essersi verificati in una determinata stringa di eventi eseguita dal sistema.

Questo è il problema che si pone l'event diagnosis.

Se gli eventi non osservabili corrispondono a fault di componenti del sistema, allora conoscere se questi eventi si siano verificati è molto importante nel monitoraggio delle performance del nostro sistema.

Da osservazioni continuative del comportamento di un sistema in termini di accadimento di eventi (osservabili per definizione), è possibile mediante l'applicazione di opportune tecniche ridurre l'incertezza riguardante la stringa di eventi eseguita dal sistema.

Ad esempio, se dopo una determinata stringa di eventi ci aspettiamo di raggiungere lo stato z, ma ci ritroviamo nello stato y, allora abbiamo la certezza che si sia verificato almeno un evento non osservabile.

Questo lavoro è compito di un automa particolare, il diagnoser.

3.1 Diagnoser

I diagnoser sono una variazione degli osservatori e vengono utilizzati per tenere traccia del comportamento del sistema e diagnosticare, se possibile, l'occorrenza degli eventi non osservabili di interesse.

Denotiamo il diagnoser costruito a partire dall'automa G con $Diag(G)$ o G_{diag}.

Come anticipato, i diagnoser sono simili agli osservatori con la differenza che vengono attribuite delle label agli stati di G. Per semplicità, assumiamo che siamo interessati con la diagnosi di un solo evento non osservabile, definiamolo $ed \in E_{uo}$. Se abbiamo la necessità di diagnosticare più eventi, possiamo ricorrere a due eventuali soluzioni: possiamo costruire un diagnoser per ogni evento da diagnosticare, oppure costruire un singolo diagnoser che sia in grado di tener traccia di tutti gli eventi di interesse contemporaneamente.

Utilizzeremo due tipi di label: N per "No, ed non si è ancora verificato" e Y per "Sì, ed si è verificato". Quando viene attribuita una label ad uno stato $x \in X$, scriveremo xN o xY come abbreviazione di (x, N) o (x, Y), rispettivamente. Richiamando la

procedura per la costruzione di $Obs(G)$ quando G possiede eventi non osservabili, le modifiche da sottoporre per la costruzione di $Diag(G)$ sono:

1. Quando si costruisce l'unobservable reach dello stato iniziale x_0 di G:
   a) Si attribuisca la label N agli stati che possono essere raggiunti da x_0 tramite stringhe non osservabili in $[E_{uo} \backslash \{ed\}]^*$, ovvero che non contengono l'evento ed;
   b) Si attribuisca la label Y agli stati che possono essere raggiunti da x_0 tramite stringhe non osservabili che contengono almeno un'occorrenza di ed;
   c) Se lo stato z può essere raggiunto sia con l'occorrenza di ed che senza l'occorrenza di ed, allora si creino due stati diversi raggiungibili in $Diag(G)$: zN e zY.
2. Quando si costruiscono gli stati raggiungibili successivi di $Diag(G)$:
   a) Si seguano le regole per la funzione delle transizioni di $Obs(G)$, ma con la modifica degli unobservable reaches descritta sopra;
   b) Si propaghi la label Y. Cioè ogni stato raggiungibile dallo stato zY dovrebbe essere etichettato con Y per indicare che l'evento ed si è verificato durante il percorso per raggiungere z e quindi nel processo di creazione di un nuovo stato raggiunto.
3. Non esistono stati marcati nel $Diag(G)$.

Quindi, $Diag(G)$ ha l'insieme degli eventi completamente osservabile E_o, è pertanto un automa deterministico e genera il linguaggio $\mathcal{L}(Diag(G)) = P[\mathcal{L}(G)]$. Ogni stato di $Diag(G)$ è un sottoinsieme di $X \times \{N, Y\}$.

Una conseguenza sostanziale nella modifica della procedura è il punto 1.c: infatti non c'è una corrispondenza biunivoca tra gli stati di $Diag(G)$ e quelli di $Obs(G)$. Esiste anche una procedura alternativa per la costruzione di $Diag(G)$ se sfruttiamo le regole di propagazione per le label degli stati. La label dello stato iniziale è sempre N. Una volta che la label N è stata aggiornata ad Y a causa di una sottostringa ed di un unobservable reach, la label rimarrà Y per tutti i futuri stati raggiungibili di G. Quindi ed determina il cambiamento della label da N ad Y, dopodiché la label rimane invariata per tutti gli stati raggiunti. Possiamo evidenziare queste regole di propagazione delle label sotto forma di automa. Consideriamo l'automa in figura 3.1.

Definiamo quest'automa come l'automa label per la diagnosi dell'evento ed e lo denotiamo con

$$A_{label} := (\{N, Y\}, \{ed\}, f_{label}, N).$$

I nomi degli stati in A_{label} contengono l'informazione delle label.

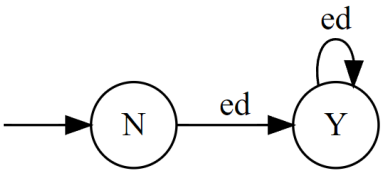

Figura 3.1: Automa A_{label}

Per ottenere $Diag(G)$, prima connettiamo G e A_{label} tramite composizione parallela, poi costruiamo l'osservatore in questa maniera:

$$Diag(G) = Obs(G \| A_{label}).$$

La composizione parallela infatti non modifica il linguaggio generato da G ma risulterà nell'attribuzione di label agli stati di G, dove gli stati saranno della forma $(x, label)$ e $label \in \{N, Y\}$.

Le label correlate ad x dipenderanno dall'assenza (N) o presenza (Y) di ed nelle stringhe che raggiungono x dallo stato iniziale x_0. La divisione degli stati si verificherà per quegli stati di G che sono raggiungibili da stringhe contenenti ed e da stringhe che non conterranno ed.

Una volta effettuata la composizione parallela, è sufficiente elaborare il suo osservatore per ottenere il diagnoser.

3.2 On-line diagnosis

On-line (o "on-the-fly") diagnosis dell'evento ed viene eseguita tenendo traccia dello stato attuale del diagnoser in risposta agli eventi osservabili dell'automa G.

Se tutti gli stati di G nello stato attuale di $Diag(G)$ hanno la label N, allora siamo sicuri che l'evento ed non si sia ancora verificato. Lo definiamo uno stato negativo.

Se tutti gli stati di G nello stato attuale di $Diag(G)$ hanno la label Y, allora siamo sicuri che l'evento ed si sia verificato ad un certo punto nel passato. In questo caso si definisce uno stato positivo. Se $t \in P[\mathcal{L}(G)]$ è stato osservato e $f_d(x_{0,d}, t)$ è positiva, allora tutte le stringhe in $P^{-1}(t) \cap \mathcal{L}(G)$ devono contenere ed; altrimenti uno stato di G in $f_d(x_{0,d}, t)$ avrebbe l'etichetta N siccome tutte le stringhe in $P^{-1}(t) \cap \mathcal{L}(G)$ conducono allo stato $f_d(x_{0,d}, t)$ di $Diag(G)$.

Infine, se lo stato attuale di $Diag(G)$ contiene almeno uno stato di G con la label N ed almeno uno stato di G con la label Y, allora l'evento ed potrebbe o non potrebbe essersi verificato nel passato. In questo caso si tratta di uno stato incerto. Infatti, esistono due stringhe in $\mathcal{L}(G)$, definiamole s_1 e s_2, tali che $P(s_1) = P(s_2)$, in cui s_1 contiene l'evento non osservabile mentre s_2 no.

3.3 Diagnosability

L'evento non osservabile ed non è diagnosticabile nel linguaggio vivo $\mathcal{L}(G)$ (per ulteriori approfondimenti sui concetti di vivezza e bloccaggio, rifarsi al libro "Introduction to discrete event systems"[1]) se esistono due stringhe sN e sY che soddisfano le seguenti condizioni:

1. sY contiene ed e sN no;
2. sY è arbitrariamente lunga dopo l'occorrenza di ed;
3. $P(sN) = P(sY)$.

Quando non esistono queste stringhe, ed si definisce diagnosticabile in $\mathcal{L}(G)$.

Onde evitare situazioni in cui G continua ad eseguire eventi non osservabili in seguito all'occorrenza di ed, quindi non viene eseguita alcuna diagnosi siccome lo stato di $Diag(G)$ non viene mai aggiornato, è solito assumere che G non abbia cicli di eventi non osservabili che siano raggiungibili dopo almeno un'occorrenza di ed.

Nel presente lavoro, manterremo questa assunzione.

Quando la proprietà di diagnosability è soddisfatta, siamo sicuri che se ed si verifica, allora $Diag(G)$ entrerà in uno stato positivo in un numero limitato di eventi dopo l'occorrenza di ed. Si osservi infatti che:

1. G non possiede cicli di eventi non osservabili dopo ed per ipotesi;
2. $Diag(G)$ non può rimanere in un ciclo di stati incerti o si contraddirebbe la diagnosability;
3. $Diag(G)$ ha un insieme finito di stati.

Quando il diagnoser ha un ciclo di stati incerti, può potenzialmente non assicurare l'occorrenza dell'evento ed per un numero arbitrariamente lungo di sottosequenze di eventi osservabili, se non lascia mai questo ciclo. Questo equivale ad una violazione della diagnosability: se troviamo due stringhe sY e sN in cui $ed \in sY$, $ed \notin sN$, $P(sN) = P(sY)$, e $P(sY)$ entra ma non lascia mai il ciclo di stati incerti nel diagnoser.

In generale, possiamo associare ad un ciclo di stati incerti in $Diag(G)$ due cicli in G, uno che include i soli stati con le label Y negli stati incerti e uno che include i soli stati con le label N negli stati incerti. Questo ciclo in $Diag(G)$ è definito un ciclo indeterminato.

Per definizione, la presenza di un ciclo indeterminato implica una violazione di diagnosability.

Invece una violazione di diagnosability implica la presenza di un ciclo indeterminato in $Diag(G)$, siccome $Diag(G)$ è un FSA. Questo ci permette di concludere che la proprietà di diagnosability può essere testata analizzando i cicli di stati incerti

in $Diag(G)$. Se uno di questi è indeterminato, allora la diagnosability non viene rispettata.

È importante sottolineare che la presenza di un ciclo di stati incerti in un diagnoser non necessariamente implica l'impossibilità di diagnosticare l'occorrenza di un evento ed con assoluta certezza.

3.4 Safe diagnosability

Per dare la definizione di "safe diagnosability" bisogna per prima cosa che $\mathcal{L}$, ovvero un linguaggio chiuso rispetto al prefisso, soddisfi due condizioni:

1. Il linguaggio $\mathcal{L}$ deve essere vivo. In altre parole significa che esiste sempre una transizione definita ad ogni stato $x \in X$, ovvero il sistema non può raggiungere un punto in cui nessun evento (transizione) è possibile;

2. Nel linguaggio $\mathcal{L}$ non deve esistere nessun ciclo di eventi non osservabili; analiticamente abbiamo che $\exists n_0 \in N \; t.c. \; \forall ust \in \mathcal{L}, s \in \Sigma_{uo}^* \Rightarrow ||s|| \leq n_0$.

La safe diagnosability applicata ad esempio ai fini dello sviluppo di sistemi di supervisione fault-tolerant, riguarda il problema di rilevare i guasti dopo il loro verificarsi, ma prima dell'esecuzione di una data serie di comportamenti proibiti, per evitare che i fault si trasformino in rotture che potrebbero causare rischi per la sicurezza. Se il sistema è safe diagnosticabile, si possono forzare azioni di riconfigurazione del sistema non appena avviene dell'identificazione del fault. Queste, se opportunamente progettate, sono in grado di evitare che il sistema esibisca comportamenti che violano la sicurezza.

Definizione

Un linguaggio $\mathcal{L}$ che soddisfa le due condizioni precedenti (1) e (2) è detto "safe diagnosable" se è diagnosticabile e se dopo un guasto, la continuazione più breve che assicura il rilevamento non contiene alcuna stringa illegale.

Analiticamente questa definizione si può scrivere attraverso due relazioni:

Condizione di diagnosability

$$(\forall i \in \Pi_f) \; (\exists n_i \in \mathbb{N}) \; (\forall s \in \Psi(\Sigma_{fi})) \; (\forall t \in \mathbf{L}/s) \; (\|t\| \geq n_i \Rightarrow \mathcal{D})$$

dove la condizione di diagnosability $\mathcal{D}$ è:

$$\omega \in P_L^{-1}[P(st)] \Rightarrow \Sigma_{fi} \in \omega;$$

Condizione di safety

$$(\forall i \in \Pi_f)\ (\forall s \in \Psi(\Sigma_{fi}))\ (\forall t \in \mathbf{L}/s)\ t.c.\ \|t\| = n_i$$

sia t_c il prefisso più breve di t t.c $\|t_c\| = n_{t_c}$

$$\Rightarrow \bar{t_c} \cap \mathcal{K}_f^i = \emptyset.$$

Capitolo 4

Software

4.1 Matlab

Matlab, o Matrix Laboratory, è un ambiente per il calcolo numerico e l'analisi statistica, sviluppato dalla MathWorks. Matlab utilizza l'omonimo linguaggio di programmazione ed è largamente impiegato per la manipolazione di matrici, l'implementazione di algoritmi, la visualizzazione di funzioni e grafici, la creazione di interfacce utente e molto altro.

È l'ambiente di lavoro principale per questo progetto; qui sono stati implementati tutti gli algoritmi che hanno permesso: lo sparser di un file ".dot", la conversione dei dati in grafi orientati e tabelle, la manipolazione e il calcolo di automi.

4.2 Graphviz

Graphviz, o Graph Visualization Software, è un programma open source avviato da AT&T Research Labs per la rappresentazione di grafi implementati in linguaggio DOT. Fornisce anche librerie per applicazioni esterne. Graphviz è un free software con licenza Eclipse Public License.

Questo programma è stato studiato per poter poi tradurre il linguaggio DOT in visualizzazione grafica degli automi, e viceversa.

4.3 Graphviz Visual Editor

Applicazione web sviluppata da Magnus Jacobsson, ideale per la visualizzazione interattiva di grafi descritti in linguaggio DOT.

Per adesso, abbiamo usufruito di questo software esterno per la rappresentazione grafica degli automi; nulla vieta di implementare questa operazione in una funzione che disegni gli automi direttamente su Matlab attraverso il metodo plot.

4.4 Software sviluppato

Ai fini di sviluppare un sistema per la diagnosi ad eventi discreti è stato necessario innanzitutto partire dall'adozione di un formalismo per la specifica di un automa in

termini testuali e la sua conversione in termini grafici.

Specifica e grafica di un automa

Il formalismo adottato è quello per la rappresentazione di grafi implementati in linguaggio DOT.

Come prima operazione occorre tradurre l'automa iniziale G in linguaggio DOT, naturalmente questa conversione occorre farla manualmente.

Nella figura 5.1 viene riportato il codice (salvato come .dot).

Una nota importante da sottolineare è la particolare struttura che si è imposta al contenuto dei file ".dot". Infatti ci sono delle regole da rispettare per l'efficace conversione del codice:

- Il primo nodo deve coincidere con lo stato iniziale e deve essere definito prima di tutti gli altri nodi, in questo caso è il nodo 1 (Riga 3 del codice). Inoltre, se il primo nodo è anche marcato, va definito sempre in quella riga;
- Nella riga successiva devono essere definiti tutti gli stati marcati, riconoscibili dall'algoritmo grazie alla forma "doublecircle"; in questo caso sono i nodi 9 e 12 (Riga 4);
- Prima di tracciare tutte le transizioni, bisogna ristabilire la forma "circle" dei nodi (Riga 5).

Una volta scritto il codice, possiamo usufruire di Graphviz Visual Editor, un software grafico esterno online per la visualizzazione dell'automa.

Per la visualizzazione da sinistra verso destra, è stato aggiunto un comando (Riga 2 del codice).

Otteniamo quindi l'automa in figura 5.2.

Successivamente, viene eseguito lo sparser del file.dot tramite un'istruzione molto potente:

```
matlabEngine='mwdot';

[status,returnCode]=system([matlabEngine ' -Tplain ' dotFile]);
```

Questo comando restituisce un character array (definito "returnCode") contenente non solo tutti i dati riguardanti l'automa, ma anche le informazioni grafiche, quali la forma, il colore, le dimensioni dei nodi, e molto altro. Inoltre, è in grado di fornire le posizioni grafiche di nodi ed archi per un'eventuale rappresentazione dell'automa. Questi dati potrebbero essere utilissimi per una futura implementazione del codice, che possa permettere la visualizzazione dell'automa direttamente su Matlab; senza l'utilizzo quindi di software grafici esterni.

I dati ottenuti tramite lo sparser, riportati in figura 5.3, vengono poi riorganizzati sotto forma di digraph, funzione di Matlab utilizzata per i grafi orientati, ma

riadattata per questo lavoro. Infatti oltre alla possibilità di restituire la matrice di adiacenza tramite il comando G.adjacency, sono state riorganizzate le informazioni sotto forma di tabulato.

Nel nostro esempio abbiamo quindi le tabelle in figura 5.4.

L'automa G viene suddiviso in 2 tabulati: G.Nodes e G.Edges.

G.Nodes contiene in ordine le label dei nodi, gli eventi attivi per ogni nodo e le label dei nodi marcati.

G.Edges contiene le label dei nodi di partenza e di arrivo per ogni transizione e la label dell'evento associato.

Costruzione del diagnoser di un automa

Input Una volta riorganizzati i dati, occorre fornire in input gli eventi non osservabili (in questo esercizio ed, u, v) e l'evento da diagnosticare ed.

Costruzione di A_{label} Per la costruzione del diagnoser, è stato utilizzato il secondo metodo proposto. Quindi, viene costruito un automa A_{label} per le label Y ed N, come nella figura 3.1.

Composizione parallela In seguito viene effettuata la composizione parallela tra l'automa in discussione G e l'automa A_{label}. Quest'operazione serve ad annettere le label Y ed N ad ogni stato dell'automa G e quindi stabilire per ogni stato se si fosse verificato o meno l'evento da diagnosticare ed.

L'automa che si ottiene dalla composizione parallela dei due automi è in figura 5.5.

Osservatore Infine viene costruito l'osservatore del nuovo automa ottenuto, dando origine al diagnoser di G rispetto all'evento ed.

Visualizzazione del diagnoser Il diagnoser ottenuto viene poi visualizzato sul software grafico esterno; l'ho riportato quindi in figura 5.6. In questa rappresentazione, risalta immediatamente la forma rettangolare dei nodi: formalismo matematico valido sia per gli osservatori sia per i diagnoser. Per cambiare forma ai nodi dell'automa, occorre modificare nel file ".dot" l'attributo riferito alla parola chiave "shape". In questo caso è stato utilizzato l'attributo "rectangle".

Le potenzialità grafiche di Graphviz sono infatti incredibili, permettendo di modificare dimensioni, forme, colori di nodi ed archi, o di manipolare il layout e la struttura del grafo in un linguaggio semplice ed intuitivo.

Capitolo 5

Esempi

5.1 Esempio didattico

Per verificare il corretto funzionamento degli algoritmi implementati su Matlab, propongo un esercizio ripreso dal testo di riferimento [1].

In realtà l'esercizio è stato leggermente modificato, infatti sono state aggiunte casistiche da esaminare, così da consolidare in maniera soddisfacente il lavoro effettuato.

Come prima operazione, è stato tradotto l'automa iniziale G in linguaggio DOT, naturalmente questa conversione occorre farla manualmente rispettando delle regole, esposte già nel capitolo precedente.

Nella figura 5.1 viene riportato il codice (salvato come .dot).

Una volta scritto il codice, possiamo usufruire di Graphviz Visual Editor, un software grafico esterno online per la visualizzazione dell'automa.

Otteniamo quindi l'automa in figura 5.2.

Successivamente, è stato eseguito lo sparser del file ".dot" tramite l'istruzione:

```
matlabEngine='mwdot';

[status,returnCode]=system([matlabEngine ' -Tplain ' dotFile]);
```

che restituisce il character array "returnCode" in figura 5.3.

I dati ottenuti tramite lo sparser sono stati poi riorganizzati sotto forma di digraph. Nel nostro esempio abbiamo quindi le tabelle in figura 5.4.

L'automa G viene suddiviso in 2 tabulati: G.Nodes e G.Edges.

G.Nodes contiene in ordine le label dei nodi, gli eventi attivi per ogni nodo e le label dei nodi marcati.

G.Edges contiene le label dei nodi di partenza e di arrivo per ogni transizione e la label dell'evento associato.

Una volta riorganizzati i dati, sono stati forniti in input gli eventi non osservabili (in questo esercizio ed, u, v) e l'evento da diagnosticare ed.

Successivamente, è stato costruito l'automa A_{label} per le label Y ed N, come nella figura 3.1.

```
digraph finite_state_machine {
    rankdir=LR;
    node [shape = circle]; 1;
    node [shape = doublecircle]; 9 12;
    node [shape = circle];
    1 -> 2 [label = "ed"];
    1 -> 4 [label = "a"];
    2 -> 3 [label = "u"];
    2 -> 5 [label = "a"];
    3 -> 7 [label = "c"];
    4 -> 5 [label = "ed"];
    4 -> 8 [label = "b"];
    5 -> 6 [label = "u"];
    5 -> 9 [label = "b"];
    6 -> 7 [label = "g"];
    7 -> 5 [label = "v"];
    7 -> 10 [label = "b"];
    8 -> 9 [label = "ed"];
    8 -> 1 [label = "d"];
    8 -> 12 [label = "v"];
    9 -> 2 [label = "d"];
    9 -> 10 [label = "u"];
    10 -> 3 [label = "d"];
    11 -> 1 [label = "d"];
    12 -> 11 [label = "ed"];
}
```

Figura 5.1: Automa iniziale G scritto in linguaggio DOT

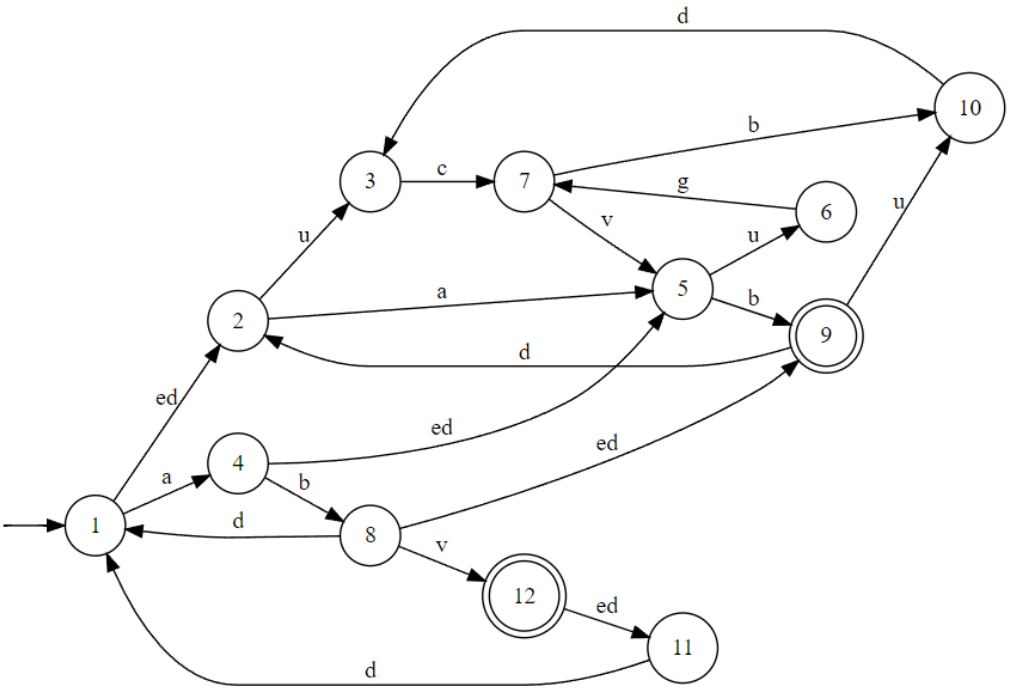

Figura 5.2: Automa G visualizzato su Graphviz Visual Editor

	1
1	graph 1.000 9.250 6.681
2	node 1 0.361 1.375 0.500 0.500 1 solid circle black lightgrey
3	node 9 7.472 3.306 0.611 0.611 9 solid doublecircle black lightgrey
4	node 12 4.597 0.958 0.622 0.622 12 solid doublecircle black lightgrey
5	node 2 1.778 3.319 0.500 0.500 2 solid circle black lightgrey
6	node 4 1.778 1.806 0.500 0.500 4 solid circle black lightgrey
7	node 3 3.111 5.069 0.500 0.500 3 solid circle black lightgrey
8	node 5 6.083 3.653 0.500 0.500 5 solid circle black lightgrey
9	node 7 4.597 5.069 0.500 0.500 7 solid circle black lightgrey
10	node 8 3.111 1.347 0.500 0.500 8 solid circle black lightgrey
11	node 6 7.472 4.750 0.500 0.500 6 solid circle black lightgrey
12	node 10 8.875 5.736 0.511 0.511 10 solid circle black lightgrey
13	node 11 6.083 0.569 0.500 0.500 11 solid circle black lightgrey
14	edge 1 2 4 0.486 1.597 0.806 2.153 1.264 2.708 1.528 3.028 ed 1.069 2.819 solid black
15	edge 1 4 4 0.597 1.444 0.819 1.514 1.167 1.611 1.417 1.694 a 1.069 1.764 solid black

Figura 5.3: Character array "returnCode" contenente tutte le informazioni grafiche dell'automa G

G.Nodes

	1 Name	2 FeasibleEvents	3 MarkedNodes
	'1'	'a,ed'	'9'
	'2'	'a,u'	'12'
	'4'	'b,ed'	''
	'9'	'd,u'	''
	'10'	'd'	''
	'12'	'ed'	''
	'11'	'd'	''
	'3'	'c'	''
	'5'	'b,u'	''
0	'8'	'd,ed,v'	''
1	'7'	'b,v'	''

G.Edges

	1 EndNodes		2 EdgeNames	3
1	'1'	'2'	'ed'	
2	'1'	'4'	'a'	
3	'2'	'3'	'u'	
4	'2'	'5'	'a'	
5	'4'	'5'	'ed'	
6	'4'	'8'	'b'	
7	'9'	'2'	'd'	
8	'9'	'10'	'u'	
9	'10'	'3'	'd'	
10	'12'	'11'	'ed'	
11	'11'	'1'	'd'	

Figura 5.4: Tabelle G.Nodes e G.Edges

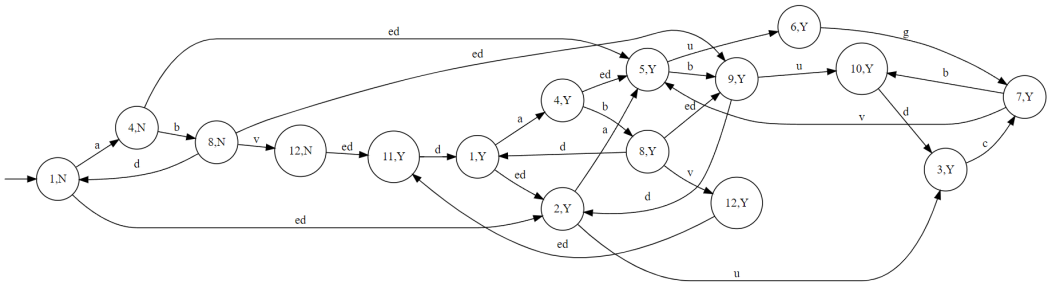

Figura 5.5: Composizione parallela tra l'automa G e l'automa A_{label}

In seguito è stata effettuata la composizione parallela tra l'automa in discussione G e l'automa A_{label}. Quest'operazione serve ad annettere le label Y ed N ad ogni stato dell'automa G e quindi stabilire per ogni stato se si fosse verificato o meno l'evento da diagnosticare *ed*. Abbiamo ottenuto quindi l'automa in figura 5.5.

Infine è stato costruito l'osservatore del nuovo automa ottenuto, dando origine al diagnoser di G rispetto all'evento *ed*.

Il diagnoser ottenuto viene poi visualizzato sul software grafico esterno; l'ho riportato quindi in figura 5.6. In questa rappresentazione, risalta immediatamente la forma rettangolare dei nodi: formalismo matematico valido sia per gli osservatori sia per i diagnoser.

5.2 Esempio pratico

Dato che gran parte del progetto è stata già sufficientemente esposta nell'esempio precedente, per snellire il testo ed alleggerire la lettura verranno evidenziate solo le parti fondamentali dell'esperimento.

L'esercizio consiste nel verificare gli algoritmi implementati su Matlab con un'applicazione pratica su di un sistema automatico. Questo problema viene ampliamente discusso dai ricercatori Andrea Paoli e Stéphane Lafortune nell'articolo "Safe diagnosability for fault-tolerant supervision of discrete-event systems" del 2005. [2]

Analizziamo ora un sistema (figura 5.7) formato da un serbatoio, una pompa e una valvola. Il primo possiede un sensore di livello mentre la tubazione che fa da collegamento può avere un sensore di flusso o di pressione. Per quanto riguarda questi ultimi due sensori, le loro uscite (outputs) sono state discretizzate con due possibili valori: nel sensore di flusso si hanno flusso (F=flow) oppure no flusso (NF=no flow); per il sensore di pressione invece pressione (P=pressure) o no pressione (NP=no pressure).

Assumiamo che l'unica modalità di guasto della valvola sia "bloccata chiusa" (stuck closed: SC).

Supponiamo che il serbatoio possa essere definito in due modalità:

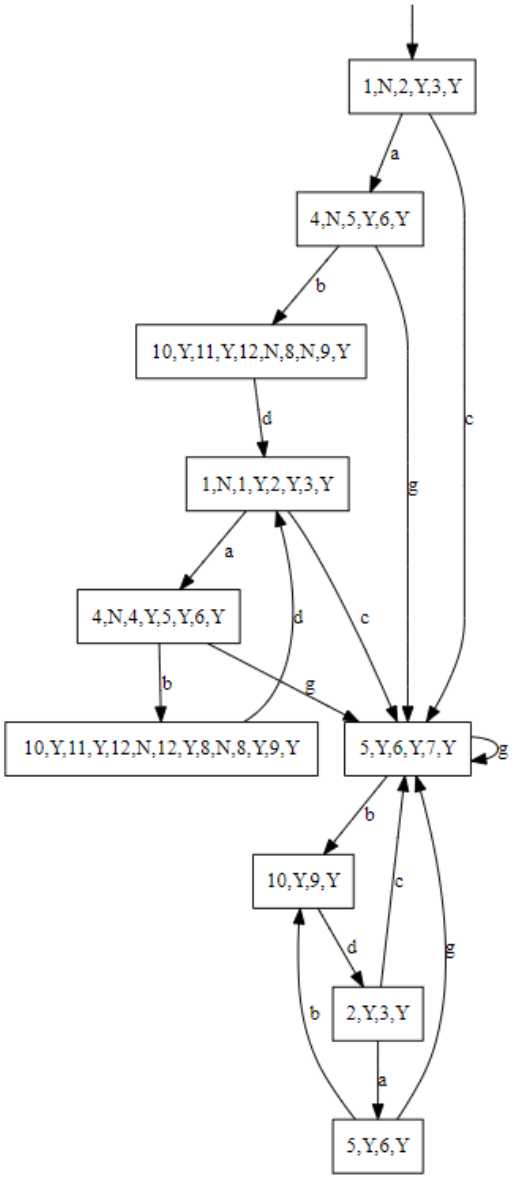

Figura 5.6: Diagnoser di G rispetto all'evento ed

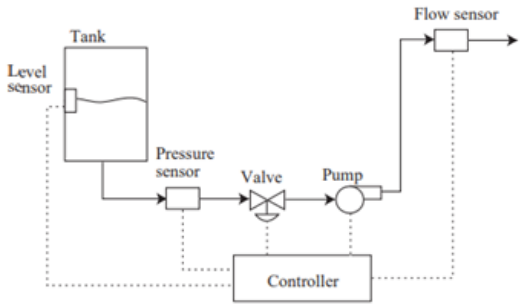

Figura 5.7: Sistema che modella un serbatoio, una pompa ed una valvola. Il controller gestisce il tutto.

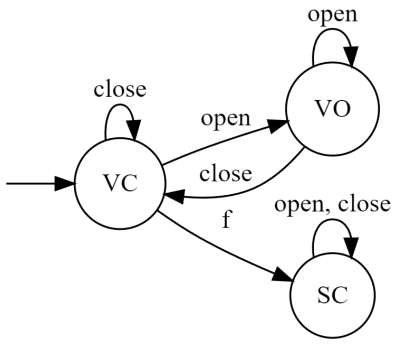

Figura 5.8: Automa che modella il comportamento della valvola

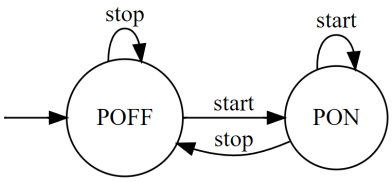

Figura 5.9: Automa che modella il comportamento della pompa

Level significa che il serbatoio è pieno e il controller risponde aprendo la valvola e attivando la pompa.

Not Level significa che il serbatoio non è pieno quindi la pompa si ferma e la valvola viene chiusa.

E che la situazione di rischio da evitare sia data da "valvola chiusa e pompa accesa".

I tre automi che modellano la valvola, la pompa e il controller, sono stati costruiti e riportati rispettivamente nelle figure 5.8, 5.9 e 5.10.

Questa tabella (figura 5.11) collega la lettura dei sensori di pressione e flusso agli stati della valvola e della pompa.

Per quanto riguarda la lettura dei sensori, dunque, la condizione di pericolo "valvola chiusa e pompa accesa" si verifica quando si ha "NF-P" ovvero nessun flusso rilevato ma pompa accesa.

Incorporando la tabella in figura 5.11 al modello di figura **??** otteniamo la macchina a stati finiti che modella il mio sistema (figura 5.12). Da questo è evidente che la

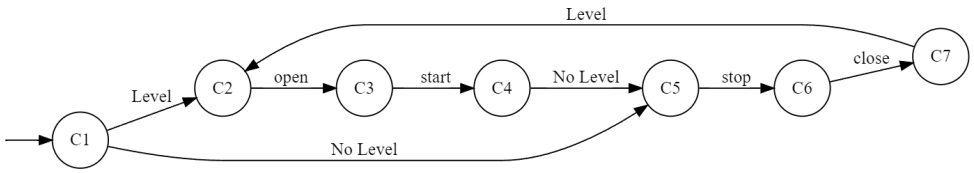

Figura 5.10: Automa che modella il comportamento del controller

	PON	POFF
VO	$F - NP$	$NF - NP$
VC	$NF - P$	$NF - P$

Figura 5.11: Tabella di lettura dei sensori e relativi stati

situazione pericolosa si verifica nello stato 12 (raggiungibile) in cui siamo nella parte "faulty behaviour" dove la valvola è bloccata chiusa "stuck closed" e la pompa è accesa "PO".

Questo significa che dato il possibile guasto della valvola, il sistema potrebbe eseguire un comportamento non sicuro, quindi bisognerebbe rilevare il guasto prima che il sistema vada nello stato 12.

Per portare a termine la diagnosi valutiamo due situazioni differenti: nel primo caso si suppone di non aver a disposizione il sensore di pressione, nel secondo caso invece è il sensore di flusso a non essere disponibile. In figura 5.13 e 5.14 sono riportati i rispettivi diagnoser.

Caso disponibilità di sensore di flusso Valutando il caso in cui ho solo il sensore di flusso si deduce che la diagnosticabilità è valida ma la "safe diagnosability" è violata nello stato 12F entrando dallo stato 3N,11F. Intuitivamente se si utilizza solo questo sensore (flow sensor) si deve aspettare che la pompa venga accesa per vedere la presenza o no di flusso: questo diagnostica il guasto ma viola la "safety condition". Infatti quando il guasto è rilevato non è ormai possibile evitare la condizione di "valvola chiusa-pompa on" che era stata identificata come condizione pericolosa e da evitare.

Caso presenza di sensore di pressione Nel caso opposto in cui disabilito il sensore di flusso e uso solo il sensore di pressione la situazione è differente perché sia la diagnosticabilità sia la safe diagnosability sono soddisfatte.

In altre parole i due casi differiscono perché in entrambi si diagnostica che il fault c'è stato ma nel primo arrivo in uno stato (12F) in cui già vedo la pericolosità del fault, mentre nel secondo mi accorgo prima quindi posso evitare di aprire la pompa e quindi posso evitare il guasto del sistema.

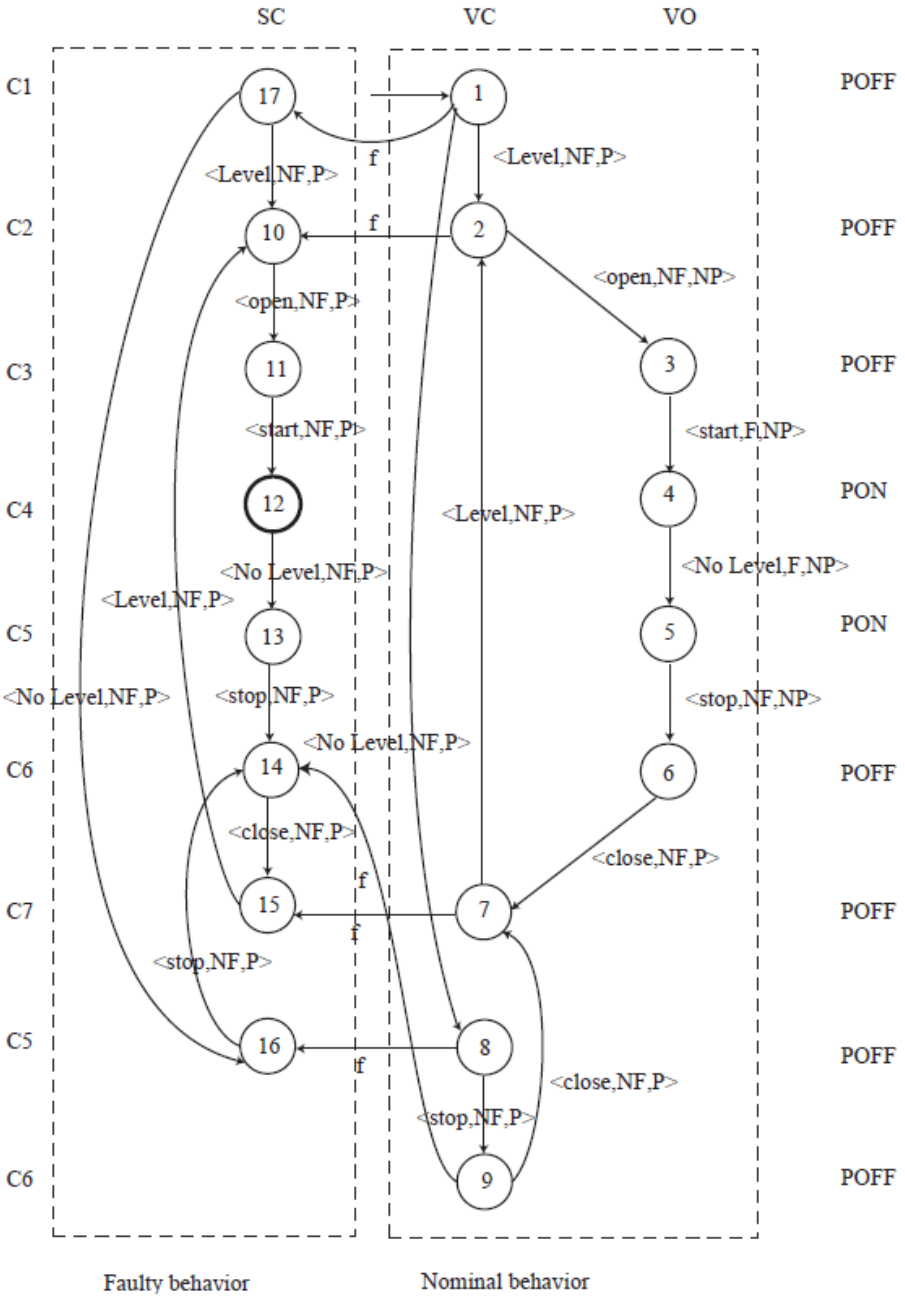

Figura 5.12: Modello finale che modella il comportamento nominale (nominal) e il comportamento del guasto (faulty) del mio sistema

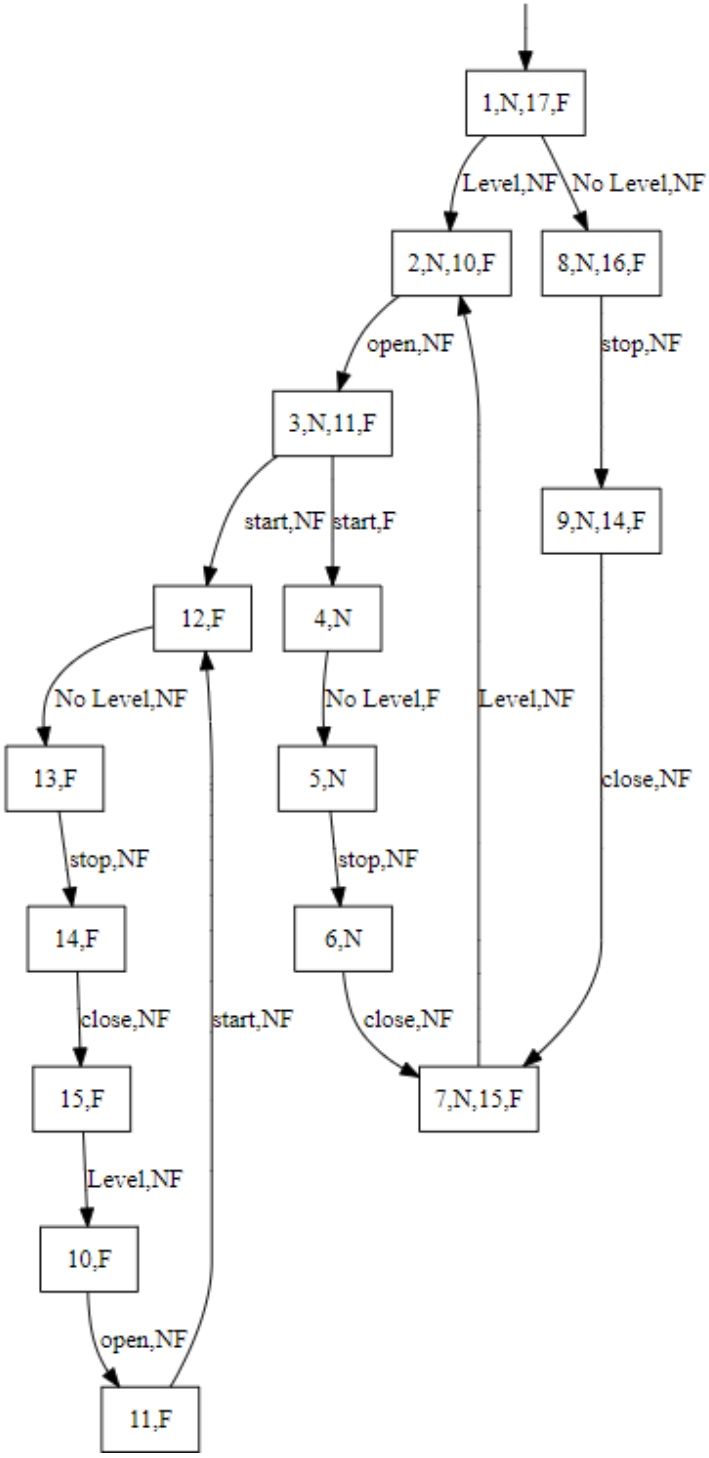

Figura 5.13: Caso diagnosticabile non sicuro

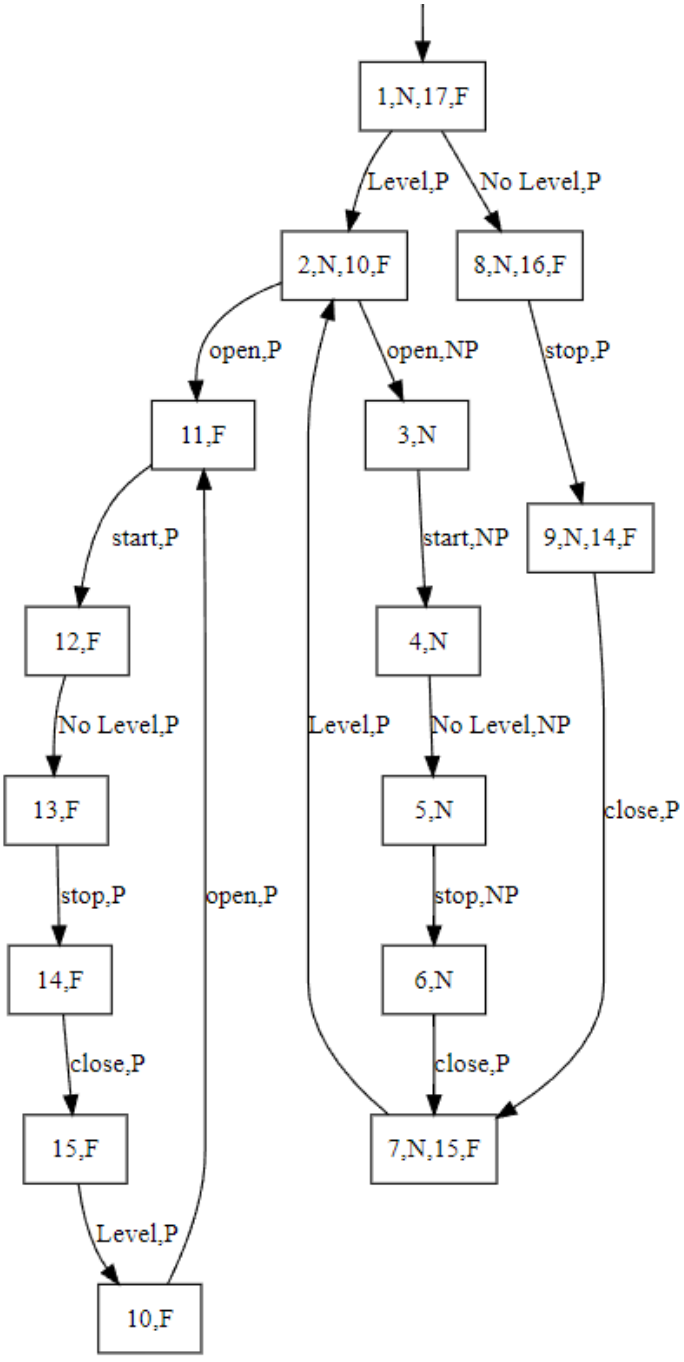

Figura 5.14: Caso diagnosticabile sicuro

Capitolo 6

Conclusioni

Questo progetto getta le basi per future implementazioni ed applicazioni.

Una possibile implementazione potrebbe essere la visualizzazione dell'automa direttamente sull'ambiente di lavoro Matlab, senza l'utilizzo di visual editor esterni.

Riguardo le applicazioni future, invece, è possibile continuare l'analisi della diagnosability di fault intermittenti utilizzando un approccio basato sulla twin-plant structure, come studiato nei recenti paper dei ricercatori Abderraouf Boussif, Baisi Liu e Mohamed Ghazel, citati in bibliografia [3, 4].

Infatti, la maggior parte della ricerca nel campo della diagnosi di fault nei sistemi ad eventi discreti si è concentrata sui fault permanenti. Tuttavia, l'esperienza con il monitoraggio di sistemi dinamici mostra che i fault intermittenti sono predominanti e che la loro diagnosi costituisce uno dei compiti più impegnativi per le attività di controllo. Tra i principali approcci esistenti per affrontare i fault permanenti, due sono stati ampiamente studiati considerando contesti diversi: l'approccio basato sul diagnoser e l'approccio basato sulla struttura Twin-plant. Quest'ultimo è stato sviluppato per far fronte ad alcuni limiti di complessità del diagnoser.

Potenzialmente, quindi, il progetto può essere esteso a qualsiasi tipo di analisi di un DES, che possa interessare l'impiego di automi.

Bibliografia

[1] Christos G Cassandras and Stéphane Lafortune. *Introduction to discrete event systems*. Springer, 2008.

[2] Andrea Paoli and Stéphane Lafortune. Safe diagnosability for fault-tolerant supervision of discrete-event systems. *Automatica*, 41(8):1335–1347, 2005.

[3] Abderraouf Boussif, Baisi Liu, and Mohamed Ghazel. A twin-plant based approach for diagnosability analysis of intermittent failures. In *2016 13th International Workshop on Discrete Event Systems (WODES)*, pages 237–244. IEEE, 2016.

[4] Abderraouf Boussif and Mohamed Ghazel. Diagnosability analysis of intermittent faults in discrete event systems using a twin-plant structure. *International Journal of Control, Automation and Systems*, 18(3):682–695, 2020.