UNIVERSITÀ
POLITECNICA
DELLE MARCHE

FACOLTÀ DI INGEGNERIA
CORSO DI LAUREA IN INGEGNERIA DELL'INFORMAZIONE

Analisi delle principali metodologie di Diagnostica di sistemi ad eventi discreti ed applicazione ad un caso di studio

Analysis of Discrete Event Systems Diagnostics methodologies and application to a case study

Relatore: Chiar.ma
Prof. Silvia Maria Zanoli

Tesi di Laurea di:
Roberto Dimitri

A.A. 2023/ 2024

Indice

Indice delle figure

Introduzione

I sistemi ad eventi discreti (DES: Descrete Event Systems) sono ED (Event Driven), ovvero sistemi la cui dinamica è guidata dall'occorrenza degli eventi stessi. In un DES lo stato cambia solo in certi istanti di tempo e con transizioni che si considerano istantanee. Ad ognuna di queste transizioni è possibile associare un evento.

Il problema principale della diagnosi di questi sistemi consiste nella rilevazione e nell'isolamento, se e quando si verifica, di un comportamento anomalo nel sistema considerato, dovuto a malfunzionamenti dei componenti, causando rischi alla sicurezza.
La diagnosticabiltà dei DES che, affronta il problema del rilevamento di eventi Fault non osservabili, partendo da un modello costruito da sequenze di eventi osservabili, è uno degli aspetti principali legati alla problematica della diagnosi guasti. Un ulteriore aspetto è quello della safe diagnosability che richiede che il rilevamento dei guasti 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 guasti 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, si sono approfonditi i concetti fondamentali della diagnostica dei DES e si sono analizzati diversi approcci presenti in letteratura, partendo dalla definizione di diagnosticabilità. I metodi proposti sono basati su sistemi modellati da automi e da reti di Petri.
Infine, si è sviluppato un esempio pratico sulla stazione presente nel laboratorio LISA (del dipartimento di Ingegneria dell'informazione dell'Università Politecnica delle Marche), costituita da un nastro trasportatore e una giostra che effettua diverse lavorazioni, utilizzando come approccio alla diagnosi il metodo basato su sistemi modellati da reti di Petri..
Il lavoro di questa tesi è organizzato nel seguente modo.
Nel capitolo 1 vengono analizzati diversi approcci alla diagnostica dei DES presenti in letteratura, che utilizzano modelli basati su automi e reti di Petri.

Nel capitolo 2 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 3 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 eventi non osservabili.

Nel capitolo 4 vengono introdotti i concetti sulle reti di Petri, la descrizione della dinamica dei DES e le sue proprietà comportamentali e strutturali, tra cui raggiungibilità, limitatezza e vivezza.

Nel capitolo 5 viene affrontato il concetto della diagnostica ad eventi, soffermandoci sulle definizioni di diagnosticabilità e sul Diagnoser che, viene utilizzato per tenere traccia del comportamento del sistema e diagnosticare, se possibile, l'occorrenza degli eventi non osservabili di interesse.

Nel capitolo 6 viene fatto un approfondimento sul metodo diagnostico basato su sistemi modellati da reti di Petri e vengono proposti degli algoritmi per implementare il Petri Net Diagnoser (PND) che viene utilizzato per la diagnosi del sistema.

Nel capitolo 7 viene proposto un esempio, utilizzando il metodo analizzato nel capitolo precedente, sul sistema presente nel laboratorio LISA.

Infine, nel capitolo 8 vengono proposte le conclusioni ed alcuni suggerimenti per eventuali progetti futuri, che estendono quanto presentato in questo lavoro di tesi.

1 Approcci alla Diagnostica ad eventi

Un sistema a eventi discreti (DES) è definito come un sistema dinamico che può essere descritto in uno spazio di stati discreti e la cui evoluzione è descritta da transizioni di stato innescate da eventi. Uno dei problemi ricorrenti di questi sistemi è l'istituzione di un'adeguata legge di controllo che garantisca la sicurezza delle attrezzature e del personale durante l'esecuzione dei compiti richiesti.
La proprietà della diagnosticabilità viene introdotta nel contesto del problema della diagnosi dei guasti. Il rilevamento e l'isolamento dei guasti sono un compito importante nel controllo automatico di sistemi complessi di grandi dimensioni. I requisiti sempre più severi in termini di prestazioni e affidabilità di sistemi complessi creati dall'uomo hanno reso necessario lo sviluppo di metodi sofisticati e sistematici per la diagnosi tempestiva e accurata dei guasti del sistema. Il problema della Fault Diagnosis ha ricevuto notevole attenzione in letteratura ed è stata proposta un'ampia varietà di schemi.
In [1] e [2], è stata introdotta la teoria della diagnosi dei guasti per sistemi ad eventi discreti modellati da automi a stati finiti (FSA). Il diagnoser proposto in [1] e [2], qui indicato come G_{diag}, può essere utilizzato per il rilevamento e l'isolamento online di eventi di guasto e anche per la verifica offline della diagnosticabilità del linguaggio generato dal sistema. Sebbene questo diagnoser possa essere facilmente implementato su un computer, tale pratica è, in generale, evitata, poiché, nel peggiore dei casi, lo spazio degli stati del diagnoser cresce esponenzialmente nella cardinalità dello spazio degli stati del modello dell'impianto G. In [1], si afferma che la diagnosi online può essere effettuata senza memorizzare tutto lo spazio degli stati di G_{diag}, essendo sufficiente ricordare solo lo stato attuale del diagnoser e, dopo il verificarsi di un evento osservabile, aggiornare il suo stato. Tuttavia, il modo esatto di effettuare tale attuazione non è dettagliato in [1].
In letteratura sono proposti diversi approcci per la diagnosi di sistemi ad eventi discreti. Di seguito vengono brevemente riportati e discussi alcuni di questi.

In [3] viene proposto un metodo per la diagnosi dei guasti che evita l'uso diretto del modello del sistema composto per l'implementazione del diagnoser, riducendo il costo computazionale per la diagnosi. Il metodo consiste nel calcolo di un diagnoser sincrono

(SD), che è costruito a partire dai modelli di comportamento privi di guasti dei componenti del sistema, e fornisce un superset (insieme che contiene tutti gli elementi di un altro insieme, possibilmente altri elementi aggiuntivi) della stima dello stato del sistema composto. Questo garantisce che il modello SD copra tutte le possibili situazioni osservabili, anche quelle che potrebbero non essere presenti nel modello senza guasti. La SD non cresce esponenzialmente con il numero di componenti del sistema e può essere facilmente implementata su un controllore logico programmabile (PLC) utilizzando la strategia proposta in [6].
Viene mostrato che una diagnosi sincrona centralizzata è un caso particolare di diagnosi sincrona decentralizzata e, pertanto, il metodo di verifica proposto può essere utilizzato per verificare entrambe le proprietà. A differenza dell'approccio diagnostico modulare, in cui il diagnoser è costruito basandosi solo sul componente in cui è modellato il guasto, nella diagnosi sincrona decentralizzata, costruiamo diagnosers locali basati sul comportamento privo di guasti di tutti i componenti del sistema. Poiché osserviamo tutti i modelli dei componenti, è possibile che un diagnoser locale, costruito sulla base di un componente che non ha l'evento di guasto, identifichi la sua insorgenza, cosa che non è possibile nel metodo di diagnosi modulare. Pertanto, i sistemi che non sono diagnosticabili in modo modulare possono essere codiagnosticati in modo sincrono.

In [4] viene proposto un approccio di sintesi di controllo sicuro di sistemi ad eventi discreti temporizzati, basato su proprietà temporizzate. Questa sintesi di controllo mira a ottenere un controllo tollerante ai guasti durante il rilevamento di un guasto del sensore attraverso il diagnoser. Un blocco decisionale decide se l'errore è tollerabile o meno. Se è tollerabile, è necessaria una riconfigurazione in cui le informazioni perse sul sensore vengono sostituite da un'informazione temporizzata. Questo controllo distribuito temporizzato si basa in primo luogo su una modellazione temporizzata di ogni elemento dell'impianto, il secondo passo consiste nel determinare i diversi controllers locali temporizzati tenendo conto dei vincoli funzionali e di sicurezza locali. Al fine di definire i controllers distribuiti temporizzati, è necessario verificare i vincoli funzionali e di sicurezza complessivi.

In [5] viene proposto un approccio diagnostico decentralizzato per eseguire la diagnosi dei sistemi di produzione. Si basa su una modellazione booleana e su diagnosi locali

basate su evento-stato. I diagnosers locali diagnosticano i guasti che violano il linguaggio delle specifiche, rappresentando il comportamento desiderato del processo monitorato in base ai comandi emessi dal controller. Il comportamento desiderato viene modellato come un modello DES booleano al fine di ottenere un modello più facile da sfruttare soprattutto nel caso di sistemi complessi dove il numero di stati è elevato. Ogni diagnoser locale osserva una parte del modello di comportamento desiderato e utilizza modelli basati su eventi e stati per dedurre il verificarsi del guasto. Riferiscono le loro decisioni di diagnosi a un coordinatore. Il coordinatore fonde queste decisioni locali al fine di ottenere una decisione globale equivalente a quella di un diagnoser centralizzato. La complessità di questo coordinatore dipende dalla complessità del processo e dai casi di conflitti decisionali tra i diagnosers locali da risolvere. Questi conflitti possono sorgere nel caso di eventi comuni tra i diagnosers locali. Se non c'è conflitto decisionale, allora il coordinatore diventa banale ed esegue un'operazione di fusione booleana.

Infine, si è analizzato una metodologia di diagnosi basata su sistemi modellati da reti di Petri. In [6] viene proposto un approccio alla rete di Petri per la diagnosi online di DES modellati da automi a stati finiti G, il cui insieme di eventi di guasto, Σ_f, può essere partizionato in diversi insiemi di eventi di guasto, ad esempio Σ_{fk}, $k = 1, ..., r$, dove r denota il numero di tipi di guasto. Il metodo proposto si basa sul calcolo di un automa composto G_C, ottenuto da G e di un automa G_{Nk}, per $k = 1, ..., r$, dove ogni automa G_{Nk} modella il comportamento non difettoso di G rispetto all'insieme di eventi di guasto Σ_{fk}. In generale, G_{Nk} ha un numero minore di stati e transizioni rispetto a G, il che riduce la complessità computazionale della diagnosi online rispetto ai metodi tradizionali che utilizzano i comportamenti normali e difettosi del sistema [1]. La tecnica di diagnosi consiste nel trovare gli stati raggiungibili di G_C dopo l'osservazione di una traccia e, in base all'insieme degli stati raggiungibili di G_C, di verificare se si è verificato o meno un guasto. A tal fine, viene proposto un diagnoser di reti di Petri (Petri Net Diagnoser: PND), ottenuto da una rete di Petri binaria in grado di stimare gli stati raggiungibili di G_C dopo l'osservazione di una traccia. Il PND fornisce la struttura per la procedura di diagnosi online, che facilita l'implementazione del codice del diagnostico su un computer.
Quest'ultima metodologia è descritta in maniera approfondita nel capitolo 6.

2 Gli Automi a stati finiti

Gli automi sono un formalismo matematico che permette di descrivere con precisione e in maniera formale il comportamento di molti sistemi ad eventi discreti. 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.

2.2 Automa deterministico

Un automa deterministico, denotato con *G*, è una sestupla

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

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.

2.2 Automa non deterministico

Nella definizione di automa deterministico, lo stato iniziale è uno stato singolo, tutte le transizioni hanno una etichetta $e \in E$ 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, 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:

- $f_{nd}: X \times E \cup \{\varepsilon\} \rightarrow 2^X$ 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.

2.3 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.

2.3.1 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 $L(G)$ e $L_m(G)$ da E^* in E_s^*, $P_s[L(G)]$ e $P_s[L_s(G)]$, 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 $K_S = L(G) \subseteq E_S^*$ e $K_{m,s} = L_m(G)$ e sia E_l un insieme di eventi più grande tale che $E_l \supset E_s$.
Sia P_s la proiezione da E_l^* in E.
Un automa che genera $P_s^{-1}(K_s)$ e marca $P_s^{-1}(P(K_{m,s})$ può essere ottenuto aggiungendo auto-anelli a tutti gli stati di G per ogni evento in $E_l \setminus E_s$.

2.3.2 Operazioni di composizione

Prodotto

Consideriamo i due automi

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

siano questi accessibili.
Il prodotto di G_1 e G_2 è l'automa

$$G_1 \times G_2 \coloneqq \mathrm{Ac}(X_1 \times X_2, E_1 \cap E_2, f, \Gamma_{1\times 2}, [\![(x)]\!]_{01}, x_{02}), X_{m1} \times X_{m2})$$

dove

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

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:

$$L(G_1 \times G_2) = L(G_1) \cap L(G_2)$$
$$L_m(G_1 \times G_2) = L_m(G_1) \cap L_m(G_2)$$

Il prodotto gode inoltre delle proprietà commutativa e associativa.
Anche nell'implementazione di questa operazione è stata fondamentale la funzione degli eventi attivi Γ_{1x2}, restringendo la ricerca degli stati raggiungibili: non si analizzano tutti gli eventi in comune ai due automi, ma solo quelli che sono attivi in entrambi gli stati in esame 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

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

dove

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

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

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

Allora otteniamo:

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

$$L_m(G_1||G_2) = P_1^{-1}[L_m(G_1)] \cap P_2^{-1}[L_m(G_2]$$

Anche la composizione parallela gode della proprietà commutativa e associativa.

3 Osservatore

Abbiamo introdotto precedentemente la classe di automi non deterministici, che differiscono dagli automi deterministici per il fatto che il codominio di f è 2^X, ovvero l'insieme potenza dello spazio di stato dell'automa, e per il fatto che è ammessa la presenza di transizioni ε.
In assenza di incertezze/imperfezioni di modellazione, l'osservazione di un modello deterministico del DES assicurerebbe piena informazione riguardo lo stato. Nella realtà occorre considerare modelli non deterministici. Si può verificare che è sempre possibile trasformare un automa non deterministico in uno deterministico equivalente, ovvero in un automa deterministico che genera e marca lo stesso linguaggio dell'automa non deterministico.
Lo spazio di stato dell'automa deterministico equivalente sarà un sottoinsieme dell'insieme potenza dello spazio di stato dell'automa non-deterministico.
L'equivalente automa deterministico ricavato a partire da un automa non-deterministico prende in 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*

$$\varepsilon R(x) \coloneqq f_{nd}^{ext}(x, \varepsilon)$$

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 $Obs(G_{nd}) = \left(X_{obs}, E, f_{obs}, x_{0,obs}, X_{m,obs}\right)$ è costruito come segue:

- Step1: Si definisce $x_{0,obs} \coloneqq \varepsilon R(x_0)$. Si imposta $X_{obs} = \{x_{0,obs}\}$.

- Step2: Per ogni $B \in Xobs$ ed $e \in E$, si definisci $f_{obs}(B,e) \coloneqq UR(\{x \in X: (\exists x_e \in B)[x \in f_{nd}(x_e,e)]\})$ 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 *Xobs*. Altrimenti se $f_{nd}(x_e,e)$non è definita per ogni $x_e \in B$, allora $f_{obs}(B,e)$ non è definita.

- Step3: Ripeti il passo 2 finché l'intera parte accessibile di *Obs* (G_{nd}) non è stata costruita.

- $X_{m,obs} \coloneqq \{B \in X_{obs}: B \cap X_m \neq \emptyset\}$.

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

- *Obs* (G_{nd}) è un automa deterministico;
- $L\big(Obs(G_{nd})\big) = L(G_{nd})$;
- $L_m\big(Obs(G_{nd})\big) = L_m(G_{nd})$;

3.1 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:

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

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

$$UR(B) = \bigcup_{x \in B} UR(x)$$

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

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

- Step1: Si definisce $x_{0,obs} \coloneqq \varepsilon R(x_0)$. Si imposta $X_{obs} = \{x_{0,obs}\}$.

- Step2: Per ogni $B \in Xobs$ ed $e \in E$, si definisci $f_{obs}(B, e) \coloneqq UR(\{x \in X : (\exists x_e \in B)[x \in f_{nd}(x_e, e)]\})$ 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 $Xobs$. Altrimenti se $f(x_e, e)$non è definita per ogni $x_e \in B$, allora $f_{obs}(B, e)$ non è definita.

- Step3: Ripeti il passo 2 finché l'intera parte accessibile di $Obs\ (G)$ non è stata costruita.

- $X_{m,obs} \coloneqq \{B \in X_{obs} : B \cap X_m \neq \emptyset\}$.

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

- $Obs\ (G)$ è un automa deterministico;
- $L\left(Obs(G)\right) = P[L(G)]$;
- $L_m\left(Obs(G)\right) = P[L_m(G)]$;
- Sia $B(t) = f_{obs}\left(x_{0,obs}, t\right)$. Allora $x \in B(t)$ se x è raggiungibile in G da una stringa in $P^{-1}(t) \cap L(G)$.

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}.

4 Reti di Petri

Le reti di Petri costituiscono una alternativa agli automi nella rappresentazione di sistemi ad eventi discreti. Una caratteristica distintiva delle Reti di Petri è il fatto che questa rappresentazione include esplicitamente le condizioni per le quali un evento può essere abilitato. Le Reti di Petri possono rappresentare in modo compatto e con raffigurazioni grafiche molto intuitive concetti come la condivisone di risorse, sincronizzazione tra processi, il succedersi asincrono di eventi e operazioni concorrenti.

Definizione (Grafo bipartito)

Un grafo di reti di Petri (o una struttura di reti di Petri) è un grafo bipartito pesato

$$N = (P, T; A, w)$$

dove

- P è un insieme finito di posti (rappresentati da cerchi)
- T è un insieme finito di transizioni (rappresentati da rettangoli)
- $A \subseteq (P \times T) \cup (T \times P)$ è l'insieme di archi dai posti alle transizioni e dalle transizioni ai posti nel grafo.

Si assume che il grafo non abbia nessun posto o transizione isolati.
Generalmente P e T sono così rappresentati:

$$P = \{p1, p2, \dots, pn\} \qquad T = \{t1, t2, \dots, tm\}$$

e un arco è espresso in una delle due seguenti forme:

$$(pi, tj) \qquad (tj, pi)$$

Nella descrizione di un grafo di reti di Petri è utile utilizzare:

- $I(t_j)$ per rappresentare l'insieme dei posti in ingresso alla transizione t_j

- $O(t_j)$ per rappresentare l'insieme dei posti in uscita dalla transizione t_j

$$I(t_j) = \{p_i \in P: (p_i, t_j) \in \mathrm{A}\}, O(t_j) = \{p_i \in P: (t_j, p_i) \in \mathrm{A}\}$$

Similmente si utilizzano *I(pi)* e *O(pi)* per descrivere le transizioni ingresso e uscita per un dato posto.

Torniamo all'interpretazione data in precedenza per gli elementi delle Reti di Petri (PN), ovvero consideriamo le transizioni come gli eventi che guidano il DES e i posti come le condizioni necessarie per il verificarsi degli eventi. In questo contesto viene naturale il bisogno di disporre di un meccanismo che indichi quando queste condizioni sono di fatto verificate e quando no. Questo meccanismo si realizza attribuendo dei "gettoni" (tokens) ai posti. Un token è qualcosa che "mettiamo in un posto" essenzialmente per indicare il fatto che la condizione descritta da quel posto è soddisfatta. Nei grafi PN un token è indicato da un pallino posizionato all'interno del cerchio del corrispondente posto. L'assegnamento di token ai posti definisce una marcatura.

Definizione (Marcatura)

Una marcatura x di un grafo $PN\ (P, T;\ A, w)$ è una funzione

$$x: P \to \aleph = \{1,2,3, \dots\}$$

Quindi la marcatura x definisce un vettore riga $\mathrm{x} = [x(p1,\ x(p2), \dots, x(pn)]$ dove *n* è il numero di posti nella PN e $x(p_i)$ indica il numero di tokens nel posto p_1.

Definizione (Grafo di reti di Petri Marcate)

Un grafo di reti di Petri Marcate è una quintupla $(P, T; A, w, x)$ dove $(P, T; A, w)$ è un grafo PN e *x* è una marcatura dell'insieme dei posti *P*; $\mathrm{x} = [x(p1,\ x(p2), \dots, x(pn)] \in \aleph^n$ è il vettore riga associato a *x*.

4.1 Descrizione della dinamica di un DES

Le definizioni viste finora non descrivono esplicitamente il meccanismo di transizione delle reti di Petri, ma la definizione di tale meccanismo è per noi di fondamentale

importanza dal momento che il nostro interesse è quello di avere uno strumento, le PN in questo caso, per poter modellare la dinamica di un DES. In pratica il meccanismo di transizione di stato è insito nella struttura stessa delle PN ma abbiamo bisogna di introdurre la nozione di transizione abilitata.

Definizione (Transizione abilitata)

Una transizione t_j in una rete di Petri si dice abilitata se:

$$x(p_i) \geq w(p_i, t_j) \; per \; ogni \; p_i \in I(t_j)$$

esempio.

Nel grafo a sinistra la transizione t_1 non è abilitata mentre in quello a destra lo è.

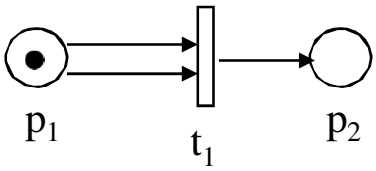

Figura 2. Transizione non abilitata

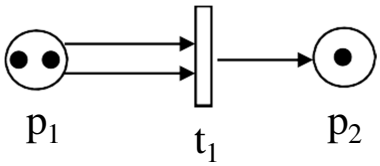

Figura 1. Transizione abilitata

In pratica, una transizione t_j si dice abilitata quando il numero di tokens in p_i è almeno grande quanto il peso dell'arco che connette p_i a t_j, per tutti i posti che sono ingressi per la transizione t_j. Dal momento che noi abbiamo associato i posti con le condizioni necessarie per il verificarsi di una transizione, allora una transizione si dice abilitata quando tutte le condizioni necessarie per il suo verificarsi sono soddisfatte.
L'insieme delle transizioni abilitate per un dato stato di una PN è equivalente all'insieme degli eventi attivi per un dato stato di un automa. Nelle Reti di Petri, il meccanismo di transizione di stato è fornito dallo spostamento dei tokens attraverso la rete e dal conseguente cambiamento di stato della rete di Petri.

Definizione (Dinamica di una rete di Petri)

La funzione di transizione di stato, $f : \aleph^n \times T \rightarrow \aleph^n$, di una rete di Petri $(P, T; A, w, x)$ per una transizione è definita se e solo se:

$$x(p_i) \geq w(p_i, t_j)\ per\ ogni\ p_i \in I(t_j) \qquad (1)$$

Se *f(x, t_j)* è definita, allora si pone *(x,t_j)* con

$$x'(p_i) = x(p_i) - w(p_i, t_j) + w(t_j, p_i) \quad i = 1, 2, \dots, n \qquad (2)$$

La condizione (1) assicura che la funzione di transizione di stato sia definita soltanto per le transizioni che sono abilitate. È importante sottolineare il fatto che il numero di tokens non deve essere necessariamente mantenuto costante dopo lo scatto di una transizione in una PN. Facendo riferimento all'equazione (2), questo è chiaro dal momento che può sicuramente risultare che:

$$\sum_{p_i \in p} w(t_j, p_i) > \sum_{p_i \in p} w(p_i, t_j) \qquad \sum_{p_i \in p} w(t_j, p_i) < \sum_{p_i \in p} w(p_i, t_j)$$

nel qual caso $x' = f(x, t_j)$ contiene tokens in numero maggiore o minore di *x*.

In generale è possibile che dopo più scatti di transizioni, lo stato risultante sia *x = [0, .., 0]*, o che il numero di tokens in uno o più posti sia cresciuto in maniera arbitrariamente grande dopo un numero arbitrariamente grande di scatti di transizioni.

Questo comportamento è la differenza fondamentale con gli automi finiti dove, per definizione si possono avere solo un numero finito di stati. Al contrario, un grafo di PN finito può corrispondere ad una rete di Petri con un numero illimitato di stati.

Un'altra importante osservazione sul comportamento dinamico di un PN è che non tutti gli stati in $\aleph^n$ possono necessariamente essere raggiunti da un PN a partire da un dato stato iniziale. Questo porta alla definizione dell'insieme degli stati raggiungibili $R[(P, T; A, w, x)]$ di un PN $(P, T; A, w, x)$. Prima di dare la definizione di stato raggiungibile abbiamo bisogno di estendere la funzione di transizione di stato *f* dal dominio $\aleph^n \times T$ al dominio $\aleph^n \times T^*$:

$f(x, \varepsilon) \coloneqq x$
$f(x, st) \coloneqq f(f(x, s), t)\ per\ s \in T^*\ e\ t \in T$

dove qui il simbolo ε deve essere interpretato come assenza di transizione che scatta.

Definizione (Stati Raggiungibili)

L'insieme degli stati raggiungibili di una PN $(P, T; A, w, x)$ è

$$R[(P, T; A, w, x)] \coloneqq \{y \in \aleph^n : \exists\, s \in t^* \ (f(x, s) = y)\}$$

La definizione sopra data assume che le transizioni abilitate scattino una alla volta. Si fa questa assunzione perché siamo interessati a tutti i possibili stati che possono essere raggiunti.

Definizione (Equazione di stato)

Dall'equazione (2) che descrive la variazione del valore dello stato per un singolo stato allo scattare di una transizione non è difficile ricavare una equazione vettoriale per ricavare il prossimo stato

$x' = [x'(p_1), x'(p_2), \dots, x'(p_n)]$ dallo stato $x = [x(p_1), x(p_2), \dots, x(p_n)]$

corrente e dall'assunzione di scatto per una particolare transizione t_j.
Definiamo innanzitutto il vettore di scatto u, della forma

$$u = [0, \dots, 0, 1, 0, \dots, 0]$$

dove l'unico 1 è nella j-esima posizione ad indicare il fatto che la j-esima transizione sta scattando.
Definiamo, inoltre, la A di una rete di Petri, come una matrice $m \times n$ le cui componenti sono della forma

$$a_{ji} = w(t_j, p_i) - w(p_i, t_j)$$

questo coincide con la differenza dei pesi w che appare in (2). A questo punto possiamo scrivere

$$x' = x + uA \qquad (3)$$

equazione che descrive il processo di transizione di stato in conseguenza di un "ingresso" u. Dunque, si ha

$$f(\mathrm{x}, t_j) = x' + uA$$

dove *f(x, t_j)* è la funzione di transizione definita prima.

L'equazione di stato fornisce un utile strumento algebrico e una alternativa ai mezzi puramente grafici per la descrizione del processo di scatto delle transizioni e cambiamento dello stato di una rete di Petri.

Finora abbiamo assunto che nelle PN le transizioni corrispondano ad eventi ma non abbiamo dato nessuna definizione precisa circa questa corrispondenza. Se vogliamo guardare ad una rete di Petri come un formalismo per descrivere un linguaggio, come abbiamo fatto con gli automi, abbiamo bisogno di specificare esattamente a quale evento corrisponda ogni transizione. Sia E l'insieme degli eventi del DES in esame, e del quale vogliamo modellare il linguaggio attraverso una rete di Petri. Si potrebbe pensare di richiedere che il modello con la PN sia tale che ad ogni transizione in *T* corrisponda un evento distinto appartenente ad *E* e viceversa. Questo però non è necessario e risulterebbe troppo restrittivo. Si pensi ad esempio come anche negli automi si possano avere due archi distinti (che partano da due stati distinti) che sono etichettati con lo stesso evento. Queste considerazioni portano a definire una Rete di Petri etichettata.

Definizione (Rete di Petri Etichettata)

Una Reti di Petri etichettata *N* è una ottupla

$$N = (P, T; A, w, E, l, x_0, X_m)$$

dove

- *(P, T ; A, w)* è una rete di Petri
- *E* è l'insieme degli eventi per le etichette delle transizioni
- $l: T \to E$ è La funzione di transizione delle etichette
- $x_0 \in \aleph^n$ è lo stato iniziale della rete (ovvero il numero iniziale di tokens in ogni posto)
- $X_m \subseteq \aleph^n$ è l'insieme di stati marcati della rete

Nelle reti di Petri l'etichetta di una transizione viene messa a fianco della transizione. La nozione di "stato marcato" in questa definizione è del tutto analoga alla nozione di stato marcato definita per gli automi. Si può definire il linguaggio marcato da una PN etichettata nel seguente modo.

Definizione (Linguaggio generato e marcato)

Il linguaggio generato da una PN etichettata $N = (P, T; A, w, E, l, x_0, X_m)$ è:

$$L(N) \coloneqq \{l(s) \in E^+ : s \in T^* \; e \; f(x_0, s) \text{ è } definita\}$$

Il linguaggio marcato da N è:

$$L_m(N) \coloneqq \{l(s) \in L(N) : s \in T^* \; e \; f(x_0, s) \in X^m\}$$

Il linguaggio *L(N)* rappresenta tutte le stringhe delle etichette associate alle transizioni che sono ottenute da tutte le possibili sequenze (finite) di scatto di transizioni in *N*, che partono dallo stato iniziale x_0 di *N*; il linguaggio marcato $L_m(N)$ è il sottoinsieme di queste stringhe che lascia la PN in uno stato che appartiene all'insieme degli stati marcati.
La classe dei linguaggi che può essere rappresentata da una PN etichettata è:

$$PNL \coloneqq \{K \subseteq E^* : \exists N = (P, T; A, w, E, l, x_0, X_m)[L_m(N) = K]\}$$

4.2 Proprietà comportamentali

4.2.1 Raggiungibilità

Come abbiamo visto la dinamica di una rete di Petri è rappresentata dallo scatto delle transizioni e dalla evoluzione dei tokens nella PN, a partire dallo stato (o marcatura) iniziale denominato con M_0. In precedenza, abbiamo dato la definizione formale di insieme degli stati raggiungibili.

Definizione (Raggiungibilità)

Il problema della raggiungibilità consiste nel verificare se esiste una sequenza di transizioni che conducano allo stato M a partire dallo stato iniziale M0, ovvero

$$M_0 \overset{\sigma}{\rightarrow} M$$

Indichiamo con $R(M_0)$ è l'insieme degli stati raggiungibili da M_0.
Essere in grado di determinare se uno stato possa essere raggiunto o meno è di per sé importante nello studio dei sistemi manifatturieri. Si pensi come talvolta risulti necessario determinare se un dato stato, richiesto, ad esempio, per fare alcune operazioni sugli utensili o per effettuare alcune misure sul prodotto, possa essere raggiunto e, nel caso, quale sia la via "migliore" per farlo.

4.2.2 Limitatezza

In un modello con PN di un sistema manifatturiero, alcuni posti rappresentano dei buffers, mentre altri posti contengono tokens che rappresentano le risorse manifatturiere. Spesso risulta utile conoscere se il numero di token in questi posti è limitato in modo da definire la dimensione del sistema o mettere in evidenza possibili errori di progettazione.

Definizione (Limitatezza e k-limitatezza di un posto)

Un posto p di una PN si dice k-limitato se il numero di tokens in quel posto non eccede mai *k*, ovvero

$$M(p) \leq k, \qquad \forall M \in R(M_0)$$

Un posto *p* di una PN si dice limitato se è *k-limitato* per qualche intero $k>0$.

Definizione (Limitatezza e k-limitatezza di una PN)

Una PN si dice *k-limitata* se il numero di tokens in qualsiasi dei suoi posti non eccede mai *k*, ovvero

$$M(p) \leq k, \qquad \forall P \in P \; e \; \forall M \in R(M_0)$$

Un a PN si dice limitata se è *k-limitata* per qualche intero $k > 0$.

Definizione (Sicurezza di PN)

Una PN si dice sicura (safe) se *1-limitata*, ovvero

$$M(p) \leq 1, \qquad \forall P \in P \; e \; \forall M \in R(M_0)$$

4.2.3 Vivezza e deadlock

In un sistema manifatturiero, spesso le operazioni vengono effettuate in parallelo. Questo richiede la sincronizzazione delle operazioni e la condivisione delle risorse perché, se così non fosse il sistema sarebbe o sottoutilizzato oppure potrebbe divenire o parzialmente o totalmente bloccato. Un sistema manifatturiero ben progettato si suppone che non cada mai in situazioni di blocco e che sia in grado di lavorare a pieno regime. La vivezza garantisce che il sistema non si blocchi mai.

Definizione (Transizioni viva)

Una transizione *t* si dice viva se da ognuno degli stati *M* raggiungibili dallo stato iniziale M_0 esiste una sequenza di scatto che mi porta in uno stato *M'* nel quale la transizione *t* è abilitata, ovvero

$$\forall M \in R(M_0) \; \exists M' \in R(M) \;\; t.c. \;\; t \text{ è } abilitata \; per \; M'$$

Definizione (PN viva)

Una PN si dice viva se tutte le sue transizioni sono vive.

Definizione (Stato di blocco Deadlock)

Uno stato *M* raggiungibile dallo stato iniziale M_0 è un blocco (deadlock) se nessuna delle transizioni della PN è abilitata.

Definizione (PN libera da deadlock)

Uno stato M raggiungibile dallo stato iniziale M_0 è un blocco (deadlock) se nessuna delle transizioni della PN è abilitata.

4.2.4 Reversibilità e stato di "home"

Eventi casuali, in un sistema manifatturiero, sono molto frequenti. Si pensi ad esempio ai possibili guasti, inceppamenti (breakdowns) delle macchine, dei sistemi di trasporto, alle rotture di attrezzi, alle rilavorazioni dovute a scarsa qualità, domande non previste, ritardi di consegna del materiale grezzo, ecc.
In casi di guasto del sistema, si deve in genere ritornare ad uno specifico stato del sistema (o ad uno degli stati tra quelli specificati dai progettisti) in maniera da permettere tutti gli aggiustamenti necessari a far ripartire il sistema. Dal punto di vista di una PN questo problema è connesso alla reversibilità e all'esistenza di stati "home".

Definizione (PN reversibile)

Una PN si dice reversibile se è possibile tornare allo stato iniziale qualunque sia lo stato raggiunto a partire dallo stato iniziale. In altre parole, una PN con stato iniziale M_0 è reversibile se

$$M_0 \in R(M) \quad \forall M \in R(M_0)$$

Definizione (Stati "home")

Uno stato M_a di una PN con stato iniziale M_0 è uno stato "home" se è raggiungibile da ogni stato raggiunto da M_0, ovvero

$$M_a \in R(M) \quad \forall M \in R(M_0)$$

Secondo queste definizioni, in una rete reversibile, ogni stato raggiungibile dallo stato iniziale è uno stato "home".

4.3 Proprietà strutturali

Le proprietà strutturali dipendono solo dalla struttura della PN e non dallo stato iniziale e dalle modalità di scatto. Dunque, queste proprietà sono molto importanti quando si deve progettare un sistema manifatturiero, dal momento che dipendono soltanto dal layout e non dal modo in cui il sistema viene gestito, cosa che a livello del progettista, non è nota. Molte delle proprietà strutturali possono essere verificate tramite tecniche algebriche. Inoltre, sotto certe condizioni, queste implicano alcune proprietà comportamentali. Abbiamo già visto il concetto di vivezza per quanto riguarda la proprietà comportamentali. Da un punto di vista strutturale la vivezza è definita nel seguente modo.

Definizione (Vivezza)

Una PN è detta strutturalmente viva se esiste uno stato iniziale M_0 tale che $PN = (N, M_0)$ è viva. Secondo questa definizione, una PN viva è anche strutturalmente viva, ma non vale il contrario. È importante notare che, a parte qualche particolare tipo di PN, è impossibile verificare la vivezza strutturale. Ciò che è possibile è ricavare condizioni necessarie da altre proprietà strutturali.

Definizione (Limitatezza)

Una PN è detta strutturalmente limitata se la PN marcata (N, M_0) è limitata per ogni stato iniziale M_0. Contrariamente a quanto avviene per la vivezza strutturale, la limitatezza strutturale richiede che il sistema rimanga limitato qualsiasi sia lo stato iniziale.

Definizione (Ripetitività)

Una PN è detta ripetitiva se esiste uno stato iniziale M_0 e una sequenza di scatto σ nella quale ogni transizione appare un numero illimitato di volte. In accordo con questa definizione, una PN strutturalmente viva è ripetitiva, ma non vale il contrario. La ripetitività è, dunque, una condizione necessaria per la vivezza strutturale e quindi, conseguentemente, è una condizione necessaria per la vivezza.

5 Diagnosi ad eventi discreti

In molte applicazioni, è interessante stabilire se gli eventi non osservabili potrebbero essersi verificati in una determinata stringa di eventi eseguita dal sistema. Questo è il problema che si pone la diagnosi di eventi discreti, descritta in [7]. 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 del Diagnoser.

5.1 Diagnoser

I Diagnoser vengono implementati come gli 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 $ed \in Euo$. 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 "*ed* non si è ancora verificato" e *Y* per "*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'unoservarble 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$[E_{uo} \setminus \{ed\}]^*$, 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.

Quindi, *Diag(G)* ha l'insieme degli eventi completamente osservabile E_o, è pertanto un automa deterministico e genera il linguaggio $L\big(Diag(G)\big) = P[L(G)]$. 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. Definiamo quest'automa come l'automa label per la diagnosi dell'evento *ed* e lo denotiamo con

$$A_{label} \coloneqq (\{N, Y\}, \{ed\}, f_{label}; Y)$$

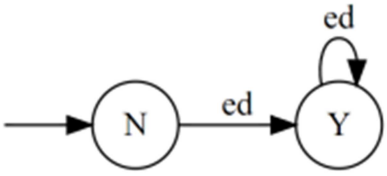

Figura 3. Automa Alabel

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.

5.2 On-line diagnosis

On-line 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 $t \in P[l(G)]$ è stato osservato e $f_d(x_{0,d}, t)$ è positiva allora tutte le stringhe in $P^{-1}(t) \cap L(G)$ devono contenere *ed*; altrimenti uno stato di *G* in$f_d(x_{0,d}, t)$ avrebbe l'etichetta *N* siccome tutte le stringhe in $P^{-1}(t) \cap L(G)$ 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 *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.

5.3 Diagnosability

Consideriamo il sistema da diagnosticare modellato come una FSM (Finite State Machine)

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

dove X è lo spazio di stato, Σ è l'insieme degli eventi, δ è la funzione di transizione parziale, e x_0 è lo stato iniziale del sistema. L'insieme degli eventi Σ è partizionato come $\Sigma = \Sigma_o \cup \Sigma_{uo}$ dove Σ_o rappresenta l'insieme degli eventi osservabili e Σ_{uo} rappresenta l'insieme degli eventi non osservabili. Sia $\Sigma_f \subseteq \Sigma$ l'insieme degli eventi Fault che devono essere diagnosticati. Partizioniamo l'insieme degli eventi di Fault in insiemi disgiunti che corrispondono a diversi tipi di guasto

$$\Sigma_f = \bigcup_{k=1}^{r} \Sigma_{fk}$$

Sia Π_f questa partizione.
Facciamo le seguenti ipotesi sul sistema in esame:

A1) *L* generato da *G* è vivo. Ciò significa che c'è una transizione definita in ogni stato x in X, cioè, il sistema non può raggiungere un punto in cui non è possibile alcun evento.
A2) Non esiste in *G* alcun ciclo di eventi non osservabile.

L'ipotesi di vivezza (A1) è fatta per motivi di semplicità, mentre l'ipotesi (A2) assicura che le osservazioni avvengano con una certa regolarità; in altre parole, richiediamo che il sistema non generi sequenze arbitrariamente lunghe di eventi non osservabili. Un linguaggio *L* è diagnosticabile se è possibile rilevare con un ritardo finito occorrenze di guasti di qualsiasi tipo utilizzando la registrazione degli eventi osservati.

Definiamo la proiezione

Si definisce $\psi(\Sigma_{fi})$ come l'insieme di tutte le tracce di L che terminano in un evento di Fault appartenente alla classe Σ_f.

Si può enunciare la seguente definizione di diagnosticabilità [1].

Definizione (Diagnosticabilità)

Un linguaggio chiuso rispetto al prefisso che soddisfa (A1) e (A2) si dice diagnosticabile rispetto alla proiezione P e rispetto alla partizione Π_f se si verifica quanto segue:

$$(\forall i \in \Pi_f)(\exists n_i \in \mathrm{N})\left(\forall \mathrm{s} \in \psi(\Sigma_{fi})\right)(\forall t \in L \backslash s)$$

$$(\|t\| \geq n_i) \Rightarrow D$$

dove la condizione di diagnosticabilità Dè:

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

La definizione di diagnosticabilità sopra citata significa quanto segue. Sia s qualsiasi traccia generata dal sistema che termina con un evento di Fault dell'insieme Σ_f, e sia t una continuazione sufficientemente lunga di s. La condizione D richiede quindi che ogni traccia appartenente al linguaggio che produce la stessa sequenza di eventi osservabili della traccia st contenga in sé un evento di Fault dell'insieme Σ_{fi}. Ciò implica che lungo ogni continuazione t di s si può rilevare il verificarsi di un guasto di tipo F_i con un ritardo finito, in particolare al massimo in n_i transizioni del sistema dopo s.

Si consideri ora il caso in cui si vuole evitare che dopo che si verifica un guasto di tipo f_i $(\mathrm{i} = 1, \ldots \mathrm{m})$ il sistema esegua una stringa proibita da un insieme finito ϕ_i, dove $\phi_i \subseteq \Sigma^*$.

Ad esempio, ciò potrebbe essere necessario per evitare che i guasti locali si trasformino in guasti che possono causare rischi per la sicurezza. Questa condizione è strettamente legata alla definizione di diagnosticabilità perché, se una stringa illegale nell'insieme ϕ_i è possibile nel linguaggio del nostro sistema, allora è necessario rilevare l'errore prima che venga eseguita una stringa illegale. Gli elementi dell'insieme ϕ_i catturano le sequenze di eventi che diventano illegali dopo il verificarsi di un guasto di tipo f_i.

Questa situazione può essere formalizzata definendo il "linguaggio illegale" $K_f^i (i = 1, \dots m)$ come segue:

$$K_f^i = \{u \in L \backslash s \; s.t. \left[s \in \psi(\Sigma_{fi})\right\} (i = 1, \dots m)$$

In altre parole, K_f^i contiene tutte le possibili continuazioni dopo un errore di tipo f_i che hanno come sottostringa una stringa proibita da ϕ_i.

Definizione (Safe Diagnosability)

Un linguaggio chiuso rispetto al prefisso che soddisfa (A1) e (A2) si dice safe diagnosable rispetto alla proiezione *P*, alla partizione Π_f e al linguaggio proibito $K_f^i (i = 1, \dots m)$ se sussistono le seguenti condizioni:

SC1) Condizione di Diagnosticabilità:

$$\left(\forall i \in \Pi_f\right)(\exists n_i \in \mathrm{N}) \left(\forall \mathrm{s} \in \psi(\Sigma_{fi})\right) (\forall t \in L \backslash s)$$

$$(\|t\| \geq n_i) \Rightarrow D$$

dove la condizione di diagnosticabilità D è:

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

SC2) Condizione di Safety:

$\left(\forall i \in \Pi_f\right)(\forall t \in L \backslash s)$ *tale che* $\|t\| = n_i$, sia t_c, $\|t_C\| = n_{t_C}$, essere il prefisso più breve di *t* tale che *D* vale, allora

$$\overline{t_C} \cap K_f^i = \emptyset$$

In parole povere, questa definizione dice che un linguaggio è diagnosticabile in modo sicuro se è diagnosticabile e se dopo un guasto, la continuazione più breve che assicura il rilevamento non contiene alcuna stringa illegale.

6 Diagnostica basata su sistemi modellati da reti di Petri

6.1 L'automa composito (G_c)

Sia G il modello automatico del sistema e sia $\Sigma_f \subseteq \Sigma_{uo}$ l'insieme degli eventi di guasto, e assumiamo che l'insieme degli eventi di guasto possa essere partizionato come

$$\Sigma_f = \bigcup_{k=1}^{r} \Sigma_{fk}$$

dove Σ_{fk} rappresenta un insieme di eventi di guasto dello stesso tipo. Sia Π_f questa partizione. Sia $L(G) = L$ il linguaggio generato da G e G_{Nk} sia il sottoautoma di G che rappresenta il comportamento non difettoso del sistema rispetto all'insieme di eventi di guasto Σ_{fk} . Supponendo che $L(G_{Nk}) = L_{Nk}$, allora L_{Nk} è un linguaggio chiuso rispetto al prefisso formato da tutte le tracce di L che non contengono alcun evento di guasto dall'insieme Σfk. Ora definiamo l'operazione proiezione $P_o: \Sigma^* \rightarrow \Sigma_o$ e $I_r = \{1,1,\ldots,r\}$. Secondo la Definizione di diagnosticabilità, L è diagnosticabile rispetto a P_o e Π_f se e solo se per tutte le tracce $st \in L \backslash L_{Nk}$ di lunghezza arbitrariamente lunga dopo il verificarsi di un evento di guasto dall'insieme Σ_{fk} , non esistono tracce $s_{Nk} \in L_{Nk}$, tali che $P_o(s_{Nk}) = P_o(st), \forall k \in Ir$.

L'automa G_C è costruito utilizzando solo il comportamento non difettoso del sistema, rispetto al tipo di guasto F_k, qui indicato come G_{Nk}.

Infine, poiché ci occupiamo della costruzione di diagnosers online, viene spontaneo fare la seguente ipotesi.

A1: Il linguaggio generato da G è diagnosticabile rispetto a P_o e Π_f.

6.1.1 Calcolo di Gc

In questo capitolo, viene presentato un algoritmo, presentato in [6], per il calcolo dell'automa G_C. A differenza degli automi diagnostici tradizionali [1], che utilizzano i comportamenti difettosi e non difettosi del sistema, l'automa G_C, come già detto precedentemente, è costruito utilizzando solo il comportamento non difettoso del

sistema, rispetto al tipo di guasto F_k, qui indicato come G_{Nk}. L'algoritmo 1 mostra la costruzione dell'automa composito G_C nel caso di più tipi di errore.

Algoritmo 1

- Step 1: Calcolare l'automa G_{Nk}, per ogni $k \in I_r$, che modella il comportamento normale di G rispetto all'insieme di eventi di guasto Σ_{fk} , come segue:
  - Step 1.1: Definire e $\Sigma_{Nk} = \Sigma \backslash \Sigma_f$.
  - Step 1.2: Costruisci l'automa A_{Nk} composto da un singolo stato N_k (anche il suo stato iniziale) con un auto-anello etichettato con tutti gli eventi in Σ_{Nk} .
  - Step 1.3: Costruisci l'automa non difettoso $G_{Nk} = G \times A_{Nk} = (Q_{Nk}, \Sigma_{Nk}, f_{Nk}, \Gamma_{Nk}, q_{0,N_k})$

- Step 2: Costruisci l'automa potenziato G^a_{Nk}, per ogni $k \in I_r$, come segue:
  - Aggiungere un nuovo stato F_k per indicare che si è verificato un evento di guasto dall'insieme Σ_{fk}.
  - Aggiungere un auto-anello in F_k con tutti gli eventi $\sigma \in \Sigma$.
  - Definisci $f(x_{Nk}, \sigma_{fk}) = F_k$ per tutti gli stati $x_{Nk} = (q, N_k) \in Q_{Nk}$ tale che $\sigma_{fk} \in \Gamma(q)$.

- Step 3: Calcola $G_c = (Q_c, \Sigma, f_c, \Gamma_c, q_{0,C}) = G^a_{N1} \| G^a_{N2} \| \| \dots \| G^a_{Nr}$.

L'idea alla base della costruzione di G_C è che: mentre il sistema è nel comportamento non difettoso, G_C segue lo stesso percorso di G; una volta che si verifica un Fault appartenente a Σ_{fk}, G_C passa a uno stato di guasto e vi rimane per sempre, indicando che si è verificata un guasto appartenente a Σ_{fk}.

Esempio 1.

Si consideri il sistema modellato dall'automa G mostrato in Figura 4, dove $\Sigma = \{a, b, c, \sigma_u, \sigma_{f1}, \sigma_{f2}\}$, $\Sigma_o = \{a, b, c\}$, $\Sigma_{uo} = \{\sigma_u, \sigma_{f1}, \sigma_{f2}\}$ e $\Sigma_f = \{\sigma_{f1}, \sigma_{f2}\}$. Si supponga

che l'insieme degli eventi Fault possa essere partizionato come $\Sigma_f = \Sigma_{f1} \cup \Sigma_{f2}$ con $\Sigma_{f1} = \{\sigma_{f1}\}$ e $\Sigma_{f2} = \{\sigma_{f2}\}$.

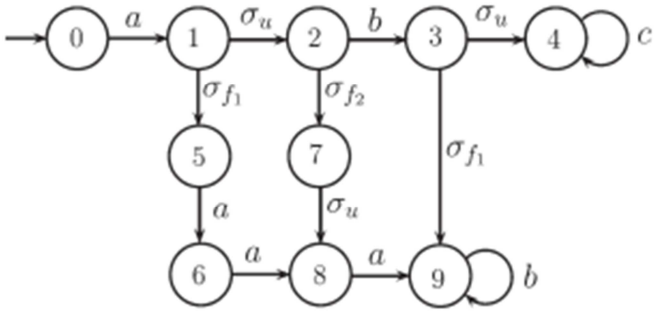

Figura 4. Automa G

Secondo l'algoritmo 1, il primo passo nella costruzione dell'automa G_C è quello di ottenere gli automi a stato singolo A_{Nk}, $k = 1, 2$, e gli automi normali (privo di Fault) $G_{Nk} = G \times A_{Nk}$. Il passo successivo è la costruzione degli automi G^a_{N1} e G^a_{N2}, mostrati rispettivamente nelle Figura 6 e Figura 5, aggiungendo gli stati difettosi F_1 e F_2 agli automi G_{N1} e G_{N2}.

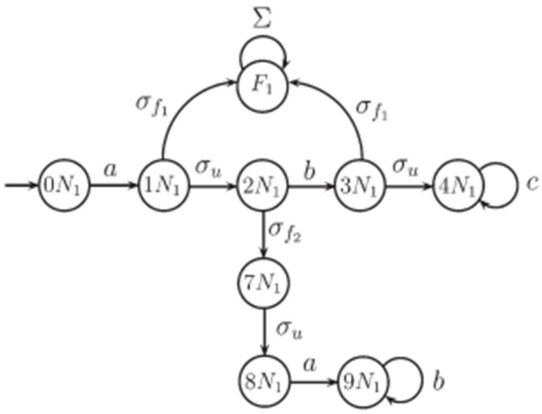

Figura 6. Automa Gn1

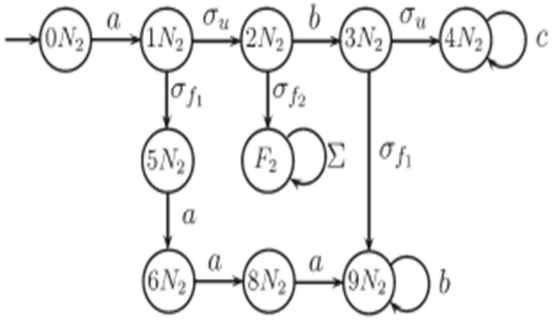

Figura 5. Automa Gn2

Il passo finale dell'Algoritmo 1 è il calcolo dell'automa $G\_c = G\_N1\text{^}a \,||G^a_{N2}$, illustrato in Figura 7.

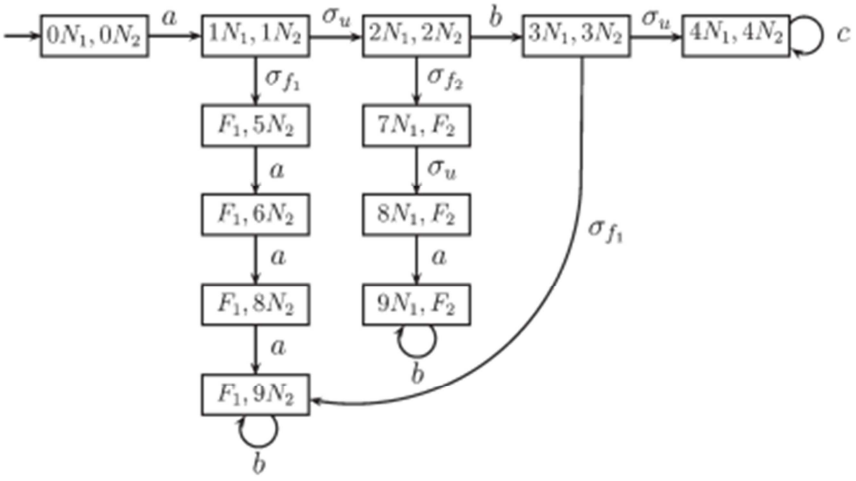

Figura 7. Automa Gc

6.1.2 Utilizzo di G_c per la diagnosi online

Al fine di mostrare come l'automa G_C possa essere utilizzato per la diagnosi online, definiremo prima una funzione che fornisce i possibili stati attuali di G_C dopo il verificarsi di un evento osservabile. Questa stima è indicata in questa tesi come *Reach(v)*, dove $v = v\sigma_o = Po(s)$ è la traccia osservata dal diagnoser dopo l'esecuzione di una traccia $s \in L$ il cui ultimo evento osservabile è σ_o, e può essere calcolata ricorsivamente come

$$Reach(\varepsilon) = UR(q_{0,C}), \qquad (6)$$

$$Reach(v\sigma_0) = UR(\delta(Reach(v), \sigma_0)) \qquad (7)$$

dove $(q_{0,C})$ è lo stato iniziale di G_C, $\delta(Reach(v), \sigma_0) = \bigcup_{i=1}^{k} \delta_C(q_{Ci}, \sigma_0)$, con $q_{Ci} \in Reach(v)$, $k = |Reach(v)|$, e $\delta_C(q_{Ci}, \sigma_0) = f_C(q_{Ci}, \sigma_0)$ se $f_C(q_{Ci}, \sigma_0)$ è definita e $\delta_C(q_{Ci}, \sigma_0) = \emptyset$, altrimenti.

Dopo l'osservazione della traccia ν, l'insieme dei possibili stati correnti di G_C, *Reach(v)*, può essere calcolato e i suoi stati possono essere utilizzati per identificare il verificarsi di un evento di guasto. Il seguente teorema fornisce le basi per il metodo diagnostico.

Teorema 1

Sia L il linguaggio generato da G e supponiamo che L sia diagnosticabile rispetto a P_o e Π_f . Sia $s \in L \setminus L_{Nk}$ tale che $\forall \omega \in L$ soddisfacenti $P_o(\omega) = P_o(s)$, $\omega \in L \setminus L_{Nk}$. Quindi, la *k-esima* coordinata di tutti i possibili stati correnti di G_C, raggiunta dopo il verificarsi di s, data da $Reach(P_o(s))$, è uguale a F_k.
Secondo il Teorema 1, se L è diagnosticabile rispetto a P_o e Π_f , allora è sempre possibile identificare l'occorrenza di una Fault di tipo F_k all'interno di un numero limitato di osservazioni verificando i possibili stati attuali di G_C.

Esempio 2.

Si consideri ancora l'automa composito G_C della Fig. 7. Supponiamo che la traccia difettosa $s = a\sigma_{f1}\, aa \in L\backslash L_{N1}$ sia stata eseguita dal sistema. Quindi, la traccia osservata è $\nu = P_0(s) = aaa$. Secondo il teorema 1, se non esiste una traccia $\omega \in L_{N1}$ tale che $P_o(\omega) = v$ allora tutti gli stati nell'insieme raggiungibile $Reach(v)$ devono avere la prima coordinata uguale a F_1. L'insieme raggiungibile $Reach(v)$ può essere ottenuto ricorsivamente secondo (6) e (7), come segue: $\mathrm{Reach}(\varepsilon) = \{(0N_1, 0N_2)\}$, $\mathrm{Reach}(a) = \{(1N_1, 1N_2), (2N_1, 2N_2), (F_1, 5N_2), (7N_1, F_2), (8N_1, F_2)\}$
$\mathrm{Reach}(aa)$ $\{(F_1, 6N_2), (9N_1, F_2)\}$ e $\mathrm{Reach}(aaa) = \{(F_1, 8N_2)\}$.
Poiché lo stato unico raggiunto dopo l'osservazione di $\nu = aaa$ ha la prima coordinata uguale a F_1, allora è possibile garantire che l'evento di guasto σ_{f1} si sia verificato.

6.2 Petri Net Diagnoser

Al fine di risolvere il problema di trovare i possibili stati attuali di G_C dopo l'osservazione di una traccia $v \in \Sigma_o$, è necessario calcolare un osservatore online che memorizza la stima degli stati attuali di G_C dopo il verificarsi di una traccia osservabile. Proporremo, in questa sezione, un osservatore di stato costruito utilizzando il formalismo della rete di Petri, ed esplorando la natura distribuita dello stato di una rete di Petri. Il primo passo nella costruzione di un osservatore di stato della rete di Petri consiste nell'ottenere una rete di Petri etichettata, N_C, dall'automa G_C. Questo può essere fatto facilmente associando a ogni stato q_{Ci} di G_C un posto p_{Ci} in N_C e associando a ciascun arco diretto in G_C, etichettato con $\sigma \in \Gamma_C(q_{Ci})$, una transizione t_{Cj}, etichettata

con σ, in N_C. Lo stato iniziale di N_C viene definito assegnando un token al posto di N_C associato allo stato iniziale di G_C e impostando a zero il numero di token degli altri posti. Questa procedura può essere formalizzata come segue.

Algoritmo 2

Sia $G\_C = (Q_C, \Sigma, f_C, \Gamma_C, q_{0,C}))$ l'automa composto del sistema. Quindi una rete di Petri a stati $N_C = (P_C, T_C, Pre_C, Post_C, x_{0,C}, \Sigma, l_C)$ può essere ottenuta come segue:

- Step1: Si crei un posto $p_{Ci} \in P_C$ per ogni stato $q_{Ci} \in Q_C$.

- Step2: Si crei una transizione $t_{Cj} \in T_C$ per ogni transizione $q_C = f_C(q_{Ci}, \sigma)$ definita in G_C, per tutti i $q_{Ci} \in Q_C$ e $\sigma \in \Gamma_C (q_{Ci})$ ed etichetta t_{Cj} con $l_C(t_{Cj}) = \{\sigma\}$.

- Step3: Si definiscano $Pre_C(p_{Ci}, t_{Cj}) = Post_C(t_{Cj}, p_{Ci}) = 1$ per ogni transizione $t_{Cj} \in T_C$, se la transizione $q_C = f_C(q_{Ci}, \sigma)$ è definita in G_C. Altrimenti, sia $Pre_C(p_{Ci}, t_{Cj}) = Post_C(t_{Cj}, p_C) = 0$.

- Step4:Si imposti $x_{0,C}(p_{C0}) = 1$ e $x_{0,C}(p_{Ci}) = 0$, per tutti i $p_{Ci} \in P_C \setminus \{p_{C0}\}$, dove p_{C0} denota il posto associato allo stato iniziale di G_C, $q_{0,C}$.

Una volta ottenuto N_C, il passo successivo per il calcolo dell'osservatore dello stato della rete di Petri per G_C è la creazione di nuovi archi, che collegano ogni transizione etichettata con un evento osservabile a posti specifici che corrispondono unoservable reach dei posti dopo l'attivazione di una transizione osservabile. Per fare ciò, sia $T_{Co} \subseteq T_C$ l'insieme di tutte le transizioni di N_C etichettate con eventi osservabili e correggere la funzione $Reach_T: T_{C0} \rightarrow 2^{P_C}$. L'insieme dei posti $Reach_T (t_{Cj})$, dove $t_{Cj} \in T_{Co}$, può essere calcolato come segue:

Algoritmo 3. (Calcolo di $Reach_T$ (t_{Cj}), $t_{Cj} \in T_{Co}$):

Siano *O(t)* e *O(p)* rispettivamente l'insieme di tutti i posti di uscita di t e l'insieme di tutte le transizioni di uscita di p. Inoltre, siano $O(P) = \bigcup_{p \in P} O(p)$ e $O(T) = \bigcup_{t \in T} O(t)$

- Step1: Sia $\{p_{out}\} = O(t_{Cj})$, $P'_r = \{p_{out}\}$ e $P_r = P'_r$.

- Step2: Formare l'insieme T'_u con tutte le transizioni di $[\![O(P')]\!]_r)$ associate ad eventi non osservabili. Se $T'_u = \emptyset$, $Reach_T(t_{Cj}) = P_r$ e ci si ferma.

- Step3: Impostare $P'_r = O(T'_u), P_r \leftarrow P_r \cup P'_r$, e tornare allo Step2.

Per implementare la unoservable Reach dopo l'attivazione di ogni transizione osservabile, un arco di peso uno, che collega ogni transizione $t_{Cj} \in T_{Co}$, ad ogni posto $p_{Ci} \in Reach_T (t_{Cj})$, deve essere aggiunto a N_C, generando una nuova rete di Petri N'_C. Dopodiché, tutte le transizioni etichettate con eventi non osservabili di N'_C, e i loro archi correlati, devono essere rimosse generando una nuova rete di Petri N_{Co}, le cui transizioni sono etichettate con solo eventi osservabili di Σ_o. Il calcolo di N_{Co} può essere formalizzato come segue.

Algoritmo 4.

Sia $N_C = (P_C, T_C, Pre_C, Post_C, x_{0,C}, \Sigma, l_C)$ la rete di Petri ottenuta da G_C utilizzando l'algoritmo 2. La rete di Petri
$N_{Co} = (P_C, T_{Co}, Pre_{Co}, Post_{Co}, x_{0,C}, \Sigma_o, l_{Co})$ può essere calcolata come segue.

- Step1: Sia $T_{Co} \subseteq T_C$ l'insieme di tutte le transizioni di N_C etichettate con eventi osservabili. Definisci una nuova funzione $Post_C: T_C \times P_C \rightarrow N$ tale che $Post'_C(t_{Cj}, p_{Ci}) = 1$, se $t_{Cj} \in T_{Co}$ e $p_{Ci} \in Reach_T(t_{Cj})$ e $Post'_C(t_{Cj}, p_{Ci}) = Post_C(t_{Cj}, p_{Ci})$, altrimenti.

- Step2: Si definisca le funzioni $\mathrm{Pre_{Co}}: \mathrm{P_C} \times \mathrm{T_{Co}} \rightarrow \mathrm{N}$ e $\mathrm{Post_{Co}}: \mathrm{T_{Co}} \times \mathrm{P_C} \rightarrow \mathrm{N}$ dove $Pre_{Co}(p_{Ci}, t_{Cj}) = Pre_C(p_{Ci}, t_{Cj})$ e $\mathrm{Post_{Co}}(\mathrm{t_{Cj}}, \mathrm{p_{Cl}}) = \mathrm{Post'_C}(\mathrm{t_{Cj}}, \mathrm{p_{Cl}})$per tutti $\mathrm{t_{Cj}} \in \mathrm{T_{Co}}$ e $\mathrm{p_{Ci}}, \mathrm{p_{Cl}} \in \mathrm{P_C}$.

- Step3: Si definisca una nuova funzione etichettata $\mathrm{l_{Co}}: \mathrm{T_{Co}} \rightarrow 2^{\Sigma_o}$ tale che $\mathrm{l_{Co}}(\mathrm{t_{Cj}}) = \mathrm{l_C}(\mathrm{t_{Cj}})$ per tutti $\mathrm{t_{Cj}} \in \mathrm{T_{Co}}$.

Si noti che per ottenere la stima degli stati di G_C, solo i posti che sono associati ai possibili stati attuali di G_C devono avere dei token e, dopo il verificarsi di un nuovo evento osservabile, il numero di token nei posti che non sono più possibili devono essere impostati a zero. Ciò implica che il numero di token in ogni posizione dell'osservatore dello stato della rete di Petri N_{SO} deve essere uguale a uno o zero. Pertanto, i posti sono costretti ad avere marcature binarie. Questo requisito può essere soddisfatto utilizzando reti di Petri binarie.

Definizione (Rete di Petri Binaria)

Una rete di Petri binaria può essere definita, come descritto in [6], come una rete di Petri con una diversa regola di evoluzione per i posti marcati raggiunti dopo l'attivazione di una transizione t_j data da

$$\bar{x}(p_i) = \begin{cases} 0, & se\ x(p_i) - Pre(p_i, t_j) + Post(t_j, p_i) = 0 \\ 1, & se\ x(p_i) - Pre(p_i, t_j) + Post(t_j, p_i) > 0 \end{cases}$$

per i=1, …, n.

È importante notare che definire N_{Co} come una rete di Petri binaria non è una condizione sufficiente per garantire che possa essere utilizzato come osservatore degli stati. Quindi, è necessario aggiungere un arco che colleghi ogni posto p_{Ci} di N_{Co} a una nuova transizione, etichettata con gli eventi osservabili di Σ_o che non sono nell'insieme di eventi attivi dello stato q_{Ci} di G_C associato a p_{Ci}. L'insieme formato con le nuove transizioni create per eliminare il token dei posti che non sono associati agli stati della stima di stato di G_C sarà indicato come l'insieme di transizione osservabile complementare, T_{Co}. Questa modifica, insieme al fatto che l'osservatore degli stati della rete di Petri è una rete di Petri binaria, garantisce che, se il posto p_{Ci} non è associato a un

possibile stato attuale di G_C dopo l'attivazione di una transizione osservabile, allora il numero di token di p_{Ci} sarà uguale a zero.

Per definire lo stato iniziale di N_{SO}, assegniamo un token a ciascun posto associato a uno stato di *UR* ($q_{0,C}$) e impostiamo il numero di token degli altri posti a zero. Infine, le transizioni auto-anello e gli archi associati possono essere rimossi dalla rete di Petri poiché l'attivazione di una transizione auto-anello non altera la stima degli stati. L'algoritmo seguente mostra come l'osservatore degli stati della rete di Petri N_{SO} può essere calcolato da N_{Co}.

Algoritmo 5.

Sia $N_{Co} = (P_C, T_{Co}, Pre_{Co}, Post_{Co}, x_{0,C}, \Sigma_o, l_{Co})$ la rete di Petri binaria ottenuta precedentemente seguendo l'algoritmo 4.

Quindi l'osservatore binario degli stati della rete di Petri $N_{so} = (P_C, T_{so}, Pre_{so}, Post_{so}, x_{0,so}, \Sigma_o, l_{so})$ può essere calcolato come segue

- Step1: Sia $T'_{Co} = \emptyset$. Per ogni $q_{Ci} \in Q_C$ tale che $\Gamma_C(q_{Ci}) \cap \Sigma_o \neq \Sigma_o$, creare una nuova transizione t_C^i e sia $T'_{Co} = T'_{Co} \cup \{t_C^i\}$.

- Step2: Sia $T_{SO} = T_{Co} \cup T'_{Co}$.

- Step3: Si definisca la nuova funzione etichettata $l_{SO}: T_{SO} \to 2^{\Sigma_o}$ dove $l_{SO}(t_{Cj}) = l_{Co}(t_{Cj})$, se $t_{Cj} \in T_{Co}$ e $l_{SO}(t_C^i) = \Sigma_o \backslash (\Gamma_C(q_{Ci}) \cap \Sigma_o)$, se $t_C^i \in T'_{Co}$.

- Step4: Si definiscano $Pre_{SO}: P_C \times T_{SO} \to N$ e $Post_{SO}: T_{SO} \times P_C \to N$ dove $Pre_{SO}(p_{Ci}, t_{Cj}) = Pre_{Co}(p_{Ci}, t_{Cj})$ e $Post_{SO}(t_{Cj}, p_{Cl}) = Post_{Co}(t_{Cj}, p_{Cl})$ per tutti $t_{Cj} \in T_{Co}$ e $p_{Ci}, p_{Cl} \in P_C$, e $Pre_{SO}(p_{Ci}, t_C^i) = 1$, $Pre_{SO}(p_{Cl}, t_C^i) = 0$ e $Post_{SO}(t_C^i, p_{Cl}) = Post_{SO}(t_C^i, p_{Ci}) = 0$, $t_C^i \in T_{Co}$ e $p_{Ci}, p_{Cl} \in P_C$ dove $i \neq l$.

- Step5: Si definisca lo stato iniziale di N_{SO} assegnando un token a ogni posto associato a uno stato di *UR* ($q_{0,C}$) e zero agli altri posti.

- Step6: Si ridefinisca T_{SO}, Pre_{SO} e $Post_{SO}$ eliminando le transizioni auto-anello e gli archi associati.

Una volta ottenuto N_{SO}, il Petri Net Diagnoser N_D può essere calcolato aggiungendo a N_{SO} transizioni t_{fk}, per $k = 1, ..., r$, dove aggiungiamo anche a ciascuna transizione t_{fk} un posto di input p_{Nk}, con inizialmente un token, e un posto di output p_{Fk} senza token, entrambi collegati a t_{fk} attraverso archi ordinari. Ogni transizione t_{fk} è associata alla verifica dell'accadimento di un tipo di guasto. Gli archi inibitori di peso uno vengono utilizzati per collegare ogni posto, associato a uno stato di G_C che ha una coordinata (q, N_k) alla transizione t_{fk}. Poiché l'arco inibitore consente l'abilitazione di una transizione solo quando il numero di token del suo posto di input è uguale a zero, allora t_{fk} sarà abilitato solo quando tutti i posti con una coordinata (q,N_k) non hanno token, il che implica che si è verificato un Fault dell'insieme Σ_{fk}. La transizione t_{fk} sarà etichettata con l'evento che si verifica sempre λ, per rappresentare che tfk si attiva immediatamente dopo essere stato abilitato, rimuovendo il token di posizione pNk e aggiungendo un token per posizionare p_{Fk}, che indica che si è verificato un errore di tipo F_k. Possiamo ora presentare un algoritmo per la costruzione del Petri Net Diagnoser N_D.

Algoritmo 6.

Sia $N_{so} = (P_C, T_{so}, Pre_{so}, Post_{so}, x_{0,so}, \Sigma_o, l_{so})$ ottenuta secondo l'algoritmo 5. Quindi il Petri Net Diagnoser $N_D = (P_D, T_D, Pre_D, Post_D, x_{0,D}, In_D, \Sigma_o \cup \{\lambda\}, l_D)$ può essere calcolato come segue.

- Step1: Sia $T_f = \bigcup_{k=1}^{r}\{t_{fk}\}$ dove t_{fk} è una transizione creata per identificare il verificarsi di un guasto dell'insieme Σ_{fk}. $T_D = T_{SO} \cup T_f$

- Step2: Si definisca la funzione etichettata $l_D: T_D \to 2^{\Sigma_o \cup \{\lambda\}}$, dove λ denota l'evento che si verifica sempre, tale che $l_D(T_D) = l_{so}(T_{so})$ per tutti $t_D \in T_{SO}$, e $l_D(T_D) = \{\lambda\}$ per tutti $t_D \in T_f$.

- Step3: Sia $P_{NF} = \cup_{k=1}^{r}\{p_{Nk}, p_{Fk}\}$, dove p_{Nk} e p_{Fk} sono, rispettivamente, i posti di input e output della transizione t_{fk}. $P_D = P_C \cup P_{NF}$.

- Step4: Definire $Pre_D: P_D \times T_D \to N$ e $Post_D: T_D \times P_D \to N$ dove $Pre_D(p_{Ci}, t_D) = Pre_{So}(p_{Ci}, t_D)$ e $Post_D(t_D, p_D) = Post_{SO}(t_D, p_D)$ per tutti $t_D \in T_{SO}$ e $p_{Ci} \in P_C$, $Pre_D(p_{Nk}, t_{fk}) = Post_D(t_{fk}, p_{Nk}) = 1$, per tutti $t_{fk} \in T_f$, $Pre_D(p_{Ci}, t_{fk}) = Post_D(t_{fk}, p_{Ci}) = 0$, per tutti $t_{fk} \in T_f$ e $p_{Ci} \in P_C$.

- Step5: Definisci la funzione degli archi inibitori $In_D: P_D \times T_D \to \{0,1\}$, dove $In_D(p_D, t_{fk}) = 1$ per ogni posto $p_D \in P_C$ associato ad uno stato di G_C che ha coordinata (q,N_k), e $In_D(p_D, t_D) = 0$, per tutti gli altri posti $p_D \in P_D$ e transizioni $t_D \in T_D$.

- Step6: Lo stato iniziale dei posti p_{Nk} è uno e dei posti p_{fk} è zero. Tutti gli altri posti hanno la stessa condizione iniziale definita da $x_{0,SO}$.

Al fine di dimostrare che il Petri Net Diagnoser N_D può essere utilizzato per la diagnosi online, in [6] viene introdotto il seguente lemma che mostra che lo stato del Petri Net State Observer N_{SO}, raggiunto dopo l'osservazione di una sequenza di eventi $v \in \Sigma_o$, fornisce la corretta stima dello stato di G_C.

Lemma 1.

Sia $\vec{x}_{SO}$ lo stato di N_{SO} raggiunto dopo l'osservazione di una sequenza di eventi v, e sia q_{obs} lo stato di $Obs(G_C)$ raggiunto dopo l'osservazione di v. Allora, esiste un posto p_{Ci} tale che $x_{SO}(p_{Ci}) = 1$ se e solo se p_{Ci} è associato ad una coordinata q_{Ci} in q_{obs}.

Teorema 2.

Sia L il linguaggio generato da G e supponiamo che L sia diagnosticabile rispetto a P_o e Π_f. Sia $s \in L$ tale che $\forall \omega \in L$ che soddisfano $P_o(\omega) = P_o(s), \omega \in L \backslash L_{Nk}$. Allora, il numero di tokens al posto di p_{Fk} di N_D, dopo l'osservazione della sequenza $P_o(s)$ è uno.

Esempio 2.

Consideriamo di nuovo l'automa G_C mostrato in Fig. 7 e supponiamo che siamo interessati a calcolare il Petri Net Diagnoser N_D per G_C. Seguendo l'algoritmo 2, otteniamo la rete di Petri a stati, N_C, di Figura 8.

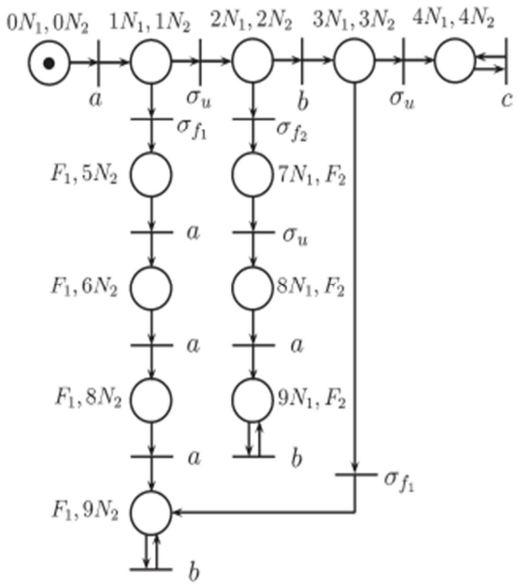

Figura 8. State Machine Petri Net Nc

In seguito, dopo l'algoritmo 4, si ottiene la rete di Petri N_{Co}, presentata in Figura 9.

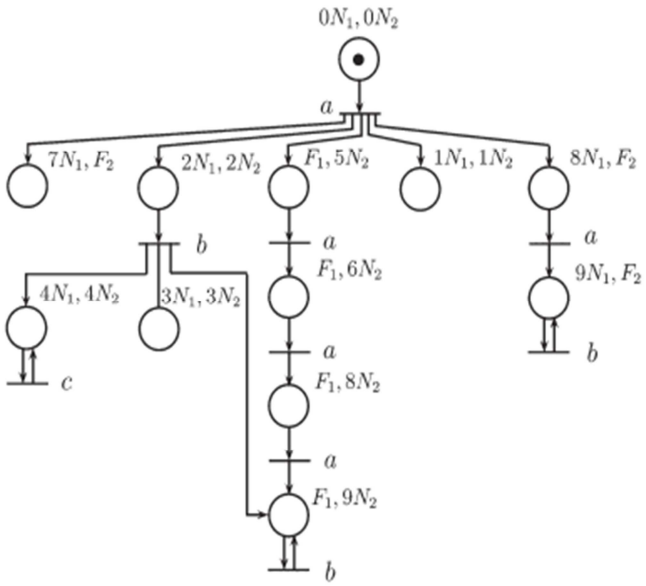

Figura 9. Binary Petri Net Nco

Quindi, secondo l'algoritmo 5, l'osservatore binario dello stato della rete di Petri N_{SO}, mostrato in Figura 10, viene ottenuto da N_{Co}.

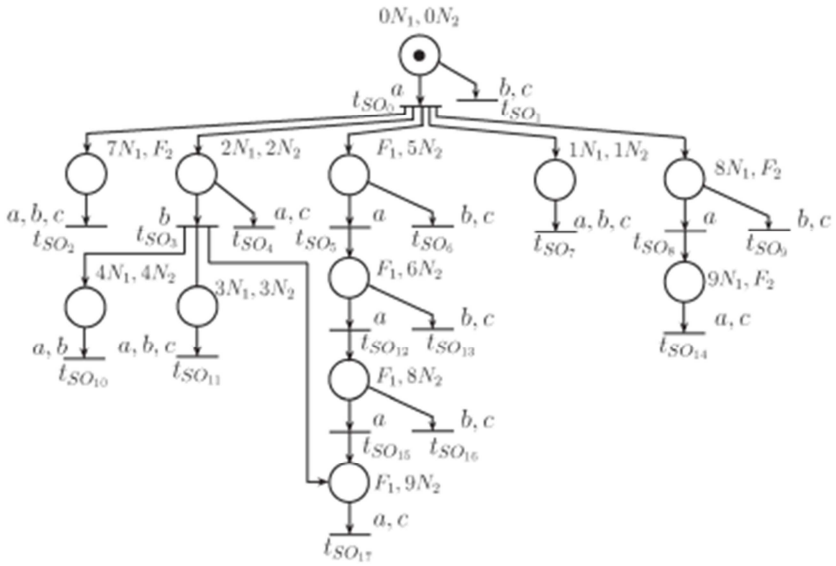

Figura 10. Petri Net State Observer Nso

Infine, seguendo l'algoritmo 6, viene calcolato il diagnoser binario della rete di Petri N_D, presentato in Figura 11.

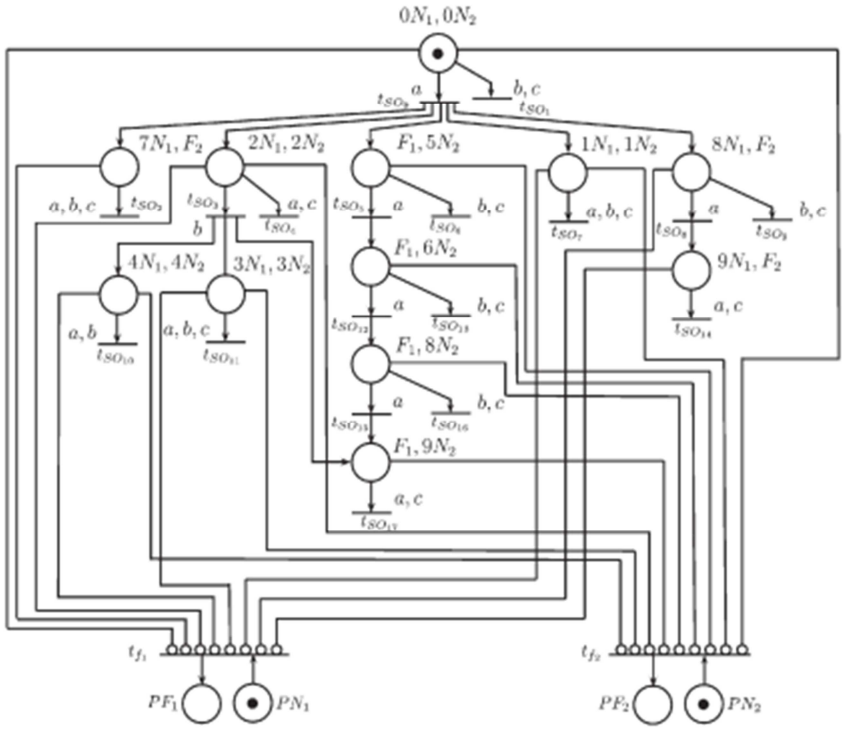

Figura 11. Petri Net Diagnoser Nd

Mostriamo ora come viene eseguita la diagnosi online utilizzando N_D. Supponiamo che la traccia difettosa $s = a\sigma f_1 aa \in L \backslash L_{N1}$ sia stata eseguita dal sistema. Quindi, la traccia osservata è $\nu = P_o(s) = aaa$. Poiché lo stato iniziale di N_D ha un token solo nel posto $(0N_1,0N_2)$, associato allo stato iniziale di G_C , si ha che dopo il verificarsi del primo evento a, la transizione t_{SO0} si attiva e l'insieme di posti associati ai possibili stati di G_C che hanno un token è dato da $\{(7N_1, F_2), (2N_1, 2N_2), (F_1, 5N_2), (1N_1, 1N_2), (8N_1, F_2)\}$. Quindi, quando si osserva il secondo evento a, le transizioni t_{SO2} , t_{SO4}, t_{SO5} , t_{SO7} e t_{SO8} scattano e l'insieme dei posti con un token è dato da $\{(F_1, 6N_2), (9N_1, F_2)\}$. Si noti che le transizioni t_{SO2}, t_{SO4} e t_{SO7} sono state create in base al passaggio 1 dell'algoritmo 5 per rimuovere il token dei posti che non sono associati a un possibile stato di G_C. Dopo il verificarsi del terzo evento a, le transizioni t_{SO12} e t_{SO14} scattano e l'unico posto di N_D che rimane con un token è $(F_1, 8N_2)$. Poiché tutti i posti associati a uno stato di G_C con una coordinata (q,N_1) non hanno token, la transizione t_{f1}, etichettata con l'evento λ, viene abilitata e si attiva, rimuovendo il token dal posto p_{N1} e aggiungendo un token a p_{F1}, indicando il verificarsi dell'evento di guasto σ_{f1} .

7 Esempio Laboratorio LISA

Dopo aver analizzato diversi approcci per la diagnosi di sistemi ad eventi discreti, si è voluto sviluppare un esempio, utilizzando uno di questi metodi, prendendo in esame due dei modelli di un sistema di produzione in scala presenti nel laboratorio LISA (Laboratory Interconnected Supervision & Automation) del Dipartimento di Ingegneria dell'informazione dell'Università Politecnica delle Marche, costituita da un nastro trasportatore e una giostra che effettua diverse lavorazioni. Quindi si è scelto di implementare un Petri Net Diagnoser (PND), descritto nel capitolo 6.2, per la diagnosi del sistema.

Per il nostro esempio si sono presi in considerazione i seguenti componenti del sistema presente in laboratorio LISA, (si veda Figura 12): Modulo Nastro Trasportatore e Modulo Giostra.

Modulo Nastro Trasportatore
Questo modulo è composto da:

- Nastro trasportatore
- Pistone
- Sensore pistone avanti
- Sensore pistone indietro
- Sensore di presenza pezzi nel dispenser
- Sensore di fine corsa nastro

Modulo Giostra
Di questo modulo si sono considerati i seguenti componenti:

- Sensore colore
- Trapano
- Pennarello
- Pistone scarto

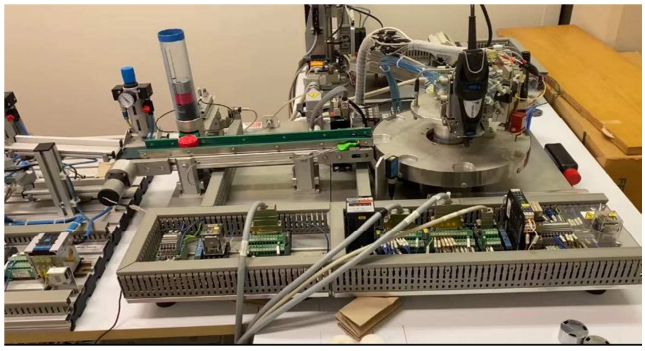

Figura 12. Stazione laboratorio LISA

Il funzionamento del sistema che si vuole implementare si basa sulla lavorazione di un pezzo: il pezzo viene rilevato dal *sensore di presenza,* il quale fa attivare il *nastro trasportatore* e il *pistone* che lo spinge sul *nastro*; i *sensori pistone avanti* e *pistone indietro* forniscono informazioni sulla posizione del *pistone*; quando il pezzo arriva alla fine del nastro il *sensore di fine corsa* ne rileva la presenza. Il pezzo continua la sua corsa e viene depositato nella prima postazione della *giostra*; il *sensore di fine corsa* attiva il movimento giostra che fa spostare il pezzo nella seconda postazione; il *sensore colore* verifica se il pezzo presente è di colore rosso; se il pezzo è di colore rosso nella terza postazione viene effettuata la lavorazione del *trapano* e nella quinta postazione non viene effettuata nessuna lavorazione; se il pezzo risulta non rosso allora nella terza postazione non viene effettuata nessuna lavorazione e invece nella quinta postazione viene effettuata la lavorazione del *pennarello*; il pezzo una volta arrivato alla sesta postazione viene scartato dal *pistone scarto*; una volta scartato, il sistema è pronto alla lavorazione del pezzo successivo. Inoltre, si considera che la giostra possa girare a causa di richieste esterne, quindi non necessariamente in presenza di un pezzo di un pezzo a fine nastro.

Nel nostro esempio si sono considerati come eventi Fault il malfunzionamento del *sensore pistone avanti* e il malfunzionamento del *sensore fine corsa*.
Nel caso in cui non ci sia nessun pezzo caricato dal nastro sulla giostra ma la giostra abbia girato, e nel caso vi sia un malfunzionamento del sensore di fine corsa non sappiamo se il pezzo sia presente o meno sulla prima postazione della giostra.
Questa casistica la si è modellata tramite un automa non deterministico.

Per tener conto di ciò nell'algoritmo 2 per l'implementazione della rete di Petri NC , nel caso di più archi in uscita da un posto dell'automa etichettati con lo stesso evento si è associata un'unica transizione etichettata con un numero di posti in uscita pari al numero di archi etichettati con lo stesso evento uscenti dal posto in esame.

Si consideri il sistema modellato dall'automa *G*, dove l'insieme degli eventi Σ e l'insieme degli eventi di Fault Σ_f risultano essere i seguenti:

$\Sigma = \{\text{start},\ \text{presenza pezzo},\ \text{nastroOn},\ \text{pistoneOn},\ \text{pistone avanti},\ \text{f1},\ \text{pistoneOff},\ \text{fine corsa},\ \text{f2},\ \text{g1},\ \text{r},\ \text{nr},\ \text{g2},\ \text{trapanoOn},\ \text{lav trapano},\ \text{g3}, \text{g4},\ \text{pen on},\ \text{lav pen},\ \text{g5},\ \text{pistone scartoOn},\ \text{pistone scartoOff}, \text{g6}\}$.

$$\Sigma_f = \{\text{f1}, \text{f2}\}$$

.dove i singoli eventi rappresentano:

- *start*: evento di *start*.
- *presenza pezzo*: evento che rappresenta l'eccitazione del *sensore di presenza*.
- *nastro On*: evento che rappresenta l'accensione del nastro.
- *pistoneOn*: evento che rappresenta l'attivazione del pistone.
- *pistone avanti*: evento che rappresenta l'eccitazione del *sensore pistone avanti*.
- *pistoneOff*: evento che rappresenta la disattivazione del pistone.
- *fine corsa*: evento che rappresenta l'eccitazione del *sensore di fine corsa* nastro.
- *f1*: evento che rappresenta il malfunzionamento del *sensore pistone avanti*.
- *f2*: evento che rappresenta il malfunzionamento del *sensore fine corsa*.
- *r*: evento che rappresenta l'eccitazione del *sensore colore*, quindi il pezzo è rosso.
- *nr*: evento che rappresenta la non eccitazione del *sensore colore*, quindi il pezzo non è rosso o non è presente.
- *trapanoOn*: evento che rappresenta l'attivazione del trapano.
- *lav trapano*: evento che rappresenta la lavorazione del trapano.
- *pen On*: evento che rappresenta l'attivazione del pennarello.
- *lav pen*: evento che rappresenta la lavorazione del pennarello.

- *pistone scartoOn*: evento che rappresenta l'attivazione pistone scarto.
- *pistone scartoOff*: evento che rappresenta la disattivazione pistone scarto.
- *g*: evento che rappresenta il movimento giostra

Si supponga che l'insieme degli eventi Fault possa essere partizionato come

$$\Sigma_f = \Sigma_{f1} \cup \Sigma_{f2} \text{ con } \Sigma_{f1} = \{f1\} \text{ e } \Sigma_{f2} = \{f2\}.$$

In seguito, nella Figura 13, Figura 14, Figura 15, Figura 16 e nella Figura 17, viene rappresentato l'automa *G*.

Nella Figura 13 viene descritto il funzionamento del Modulo Nastro Trasportatore, con il seguente insieme degli stati:

- *x0*: stato iniziale dell'automa *G*.
- *x1*: stato dopo l'evento *start.*
- *XT1*: stato che rappresenta il caso in cui non è presente nessun pezzo.
- *XT1.1*: stato che rappresenta il caso in cui non è presente nessun pezzo.
- *x2*: stato che rappresenta il caso in cui è presente un pezzo.
- *x3*: stato che rappresenta che il nastro è acceso.
- *x4*: stato che rappresenta l'azionamento del pistone.
- *x5*: stato che rappresenta il pistone è in posizione "avanti".
- *x6*: stato che rappresenta il pistone è in posizione "indietro"
- *x6.1*: stato che rappresenta il malfunzionamento del *sensore pistone avanti* (f_1).
- *P1*: stato che rappresenta la prima postazione della giostra senza che si sia verificato nessun Fault.
- *P1.1*: stato che rappresenta la prima postazione della giostra dopo il verificarsi del Fault. f_1.
- *P1.2*: stato che rappresenta la prima postazione della giostra dopo il verificarsi dei Fault. f_1 e f_2.
- *P2*: stato che rappresenta la prima postazione della giostra dopo il verificarsi del Fault. f_2.

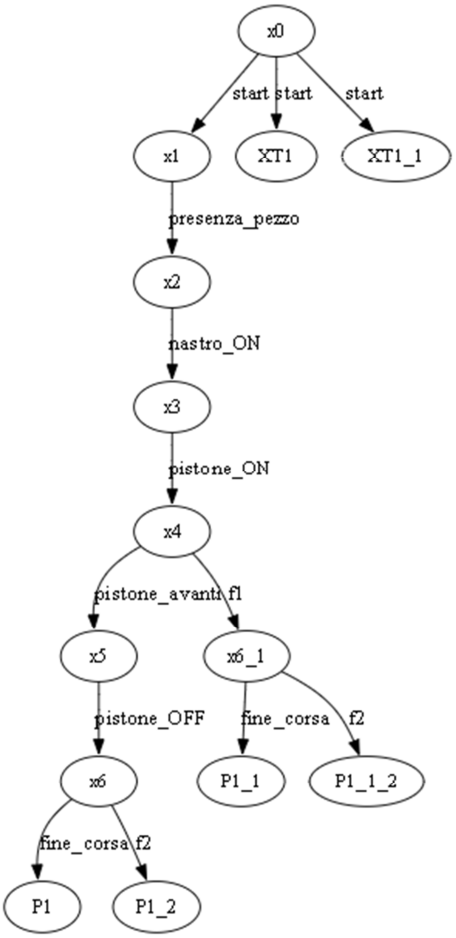

Figura 13. Automa che descrive il funzionamento nastro

Nella Figura 14 viene descritto il funzionamento del Modulo Giostra dopo l'evento $\{f_1\}$, con il seguente insieme degli stati:

- *P1.1*: stato che rappresenta la prima postazione della giostra.
- *P2.1*: stato che rappresenta la seconda postazione della giostra.
- *P2R.1*: stato che rappresenta che il pezzo è rosso e che serve successivamente ad azionare il trapano.
- *P2R_L.1*: stato che rappresenta che il pezzo è rosso.
- *P2NR_L.1*: stato che rappresenta che il pezzo non è rosso.
- *P2NR.1*: stato che rappresenta che il pezzo non è rosso e che serve successivamente a consentire il passaggio alla quarta postazione della giostra.
- *P3NT.1*: stato che rappresenta che il pezzo è rosso e che serve successivamente ad azionare il pennarello.

- *P3.1*: stato che rappresenta la terza postazione della giostra.
- *P3T.1*: stato che rappresenta l'attivazione del trapano.
- *P3T_L.1*: stato che rappresenta l'attivazione trapano e che serve successivamente a consentire il passaggio alla quarta postazione della giostra.
- *P3LT.1*: stato che rappresenta la lavorazione del trapano e che serve successivamente a consentire il passaggio alla sesta postazione della giostra.
- *P3LT_L.1*: stato che rappresenta la lavorazione del trapano e che serve successivamente a consentire il passaggio alla quarta postazione della giostra.
- *P4.1*: stato che rappresenta la quarta postazione della giostra.
- *P5.1*: stato che rappresenta la quinta postazione della giostra.
- *P5P.1*: stato che rappresenta l'attivazione del pennarello.
- *P5LP.1*: stato che rappresenta la lavorazione del pennarello.
- *P6.1*: stato che rappresenta la sesta postazione della giostra.
- *P6L.1*: stato che rappresenta l'attivazione del pistone scarto.
- *XF.1*: stato che rappresenta l'ultima postazione della giostra.

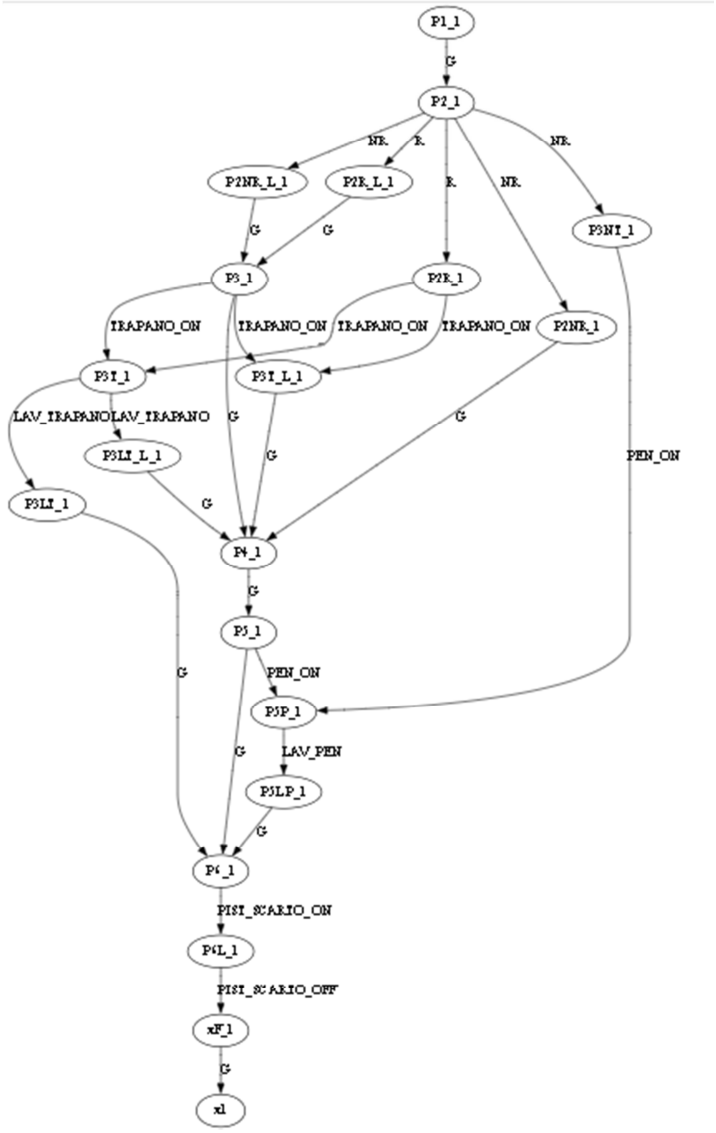

Figura 14. Automa che descrive il funzionamento della giostra dopo {f1}

Nella Figura 15, Figura 16 e nella Figura 17 viene descritto il funzionamento del Modulo Giostra dopo l'evento $\{f_1\}$, Modulo Giostra dopo l'evento $\{f_2\}$ Modulo Giostra senza il verificarsi di nessun Fault. L'insieme degli stati sono uguali a quello descritto precedentemente per la Figura 14, con la differenza della numerazione che indica quale evento di Fault si è verificato (per esempio: *P1.1.2* significa che ci troviamo nella prima postazione della giostra dopo il verificarsi degli eventi di Fault $\{f_1\}$e $\{f_2\}$).

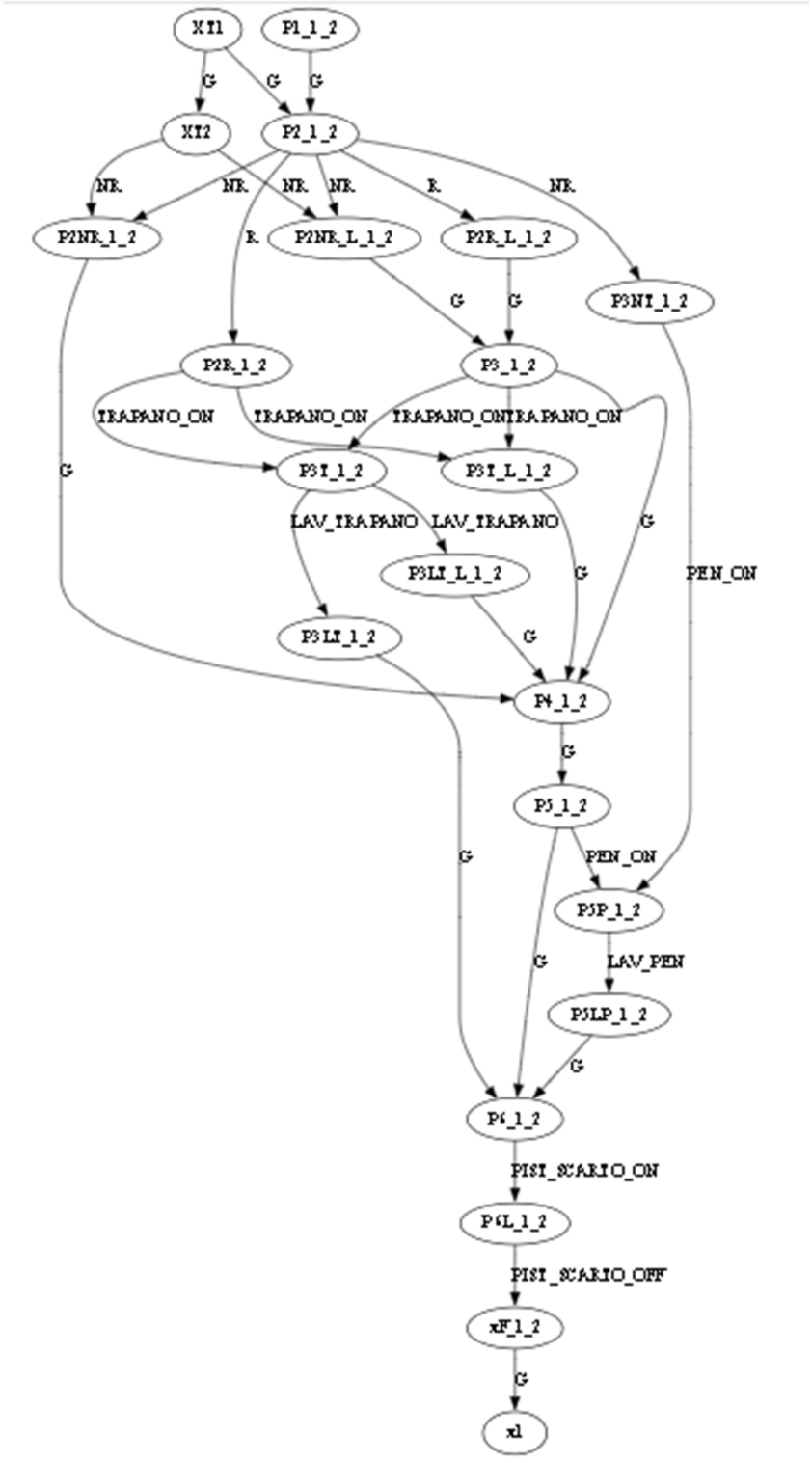

Figura 15. Automa che descrive il funzionamento della giostra dopo {f1} e {f2}

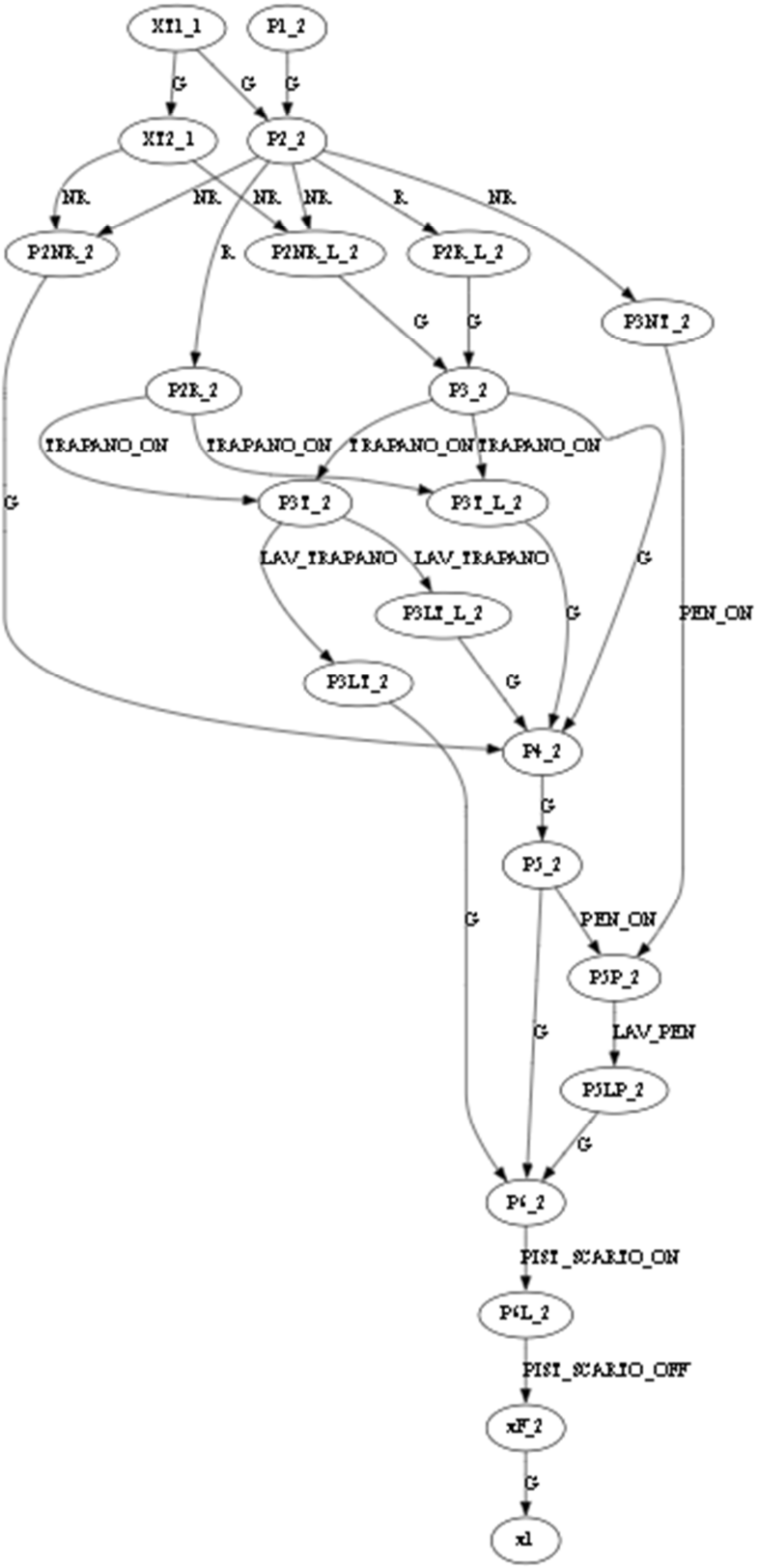

Figura 16. Automa che descrive il funzionamento della giostra dopo {f2}

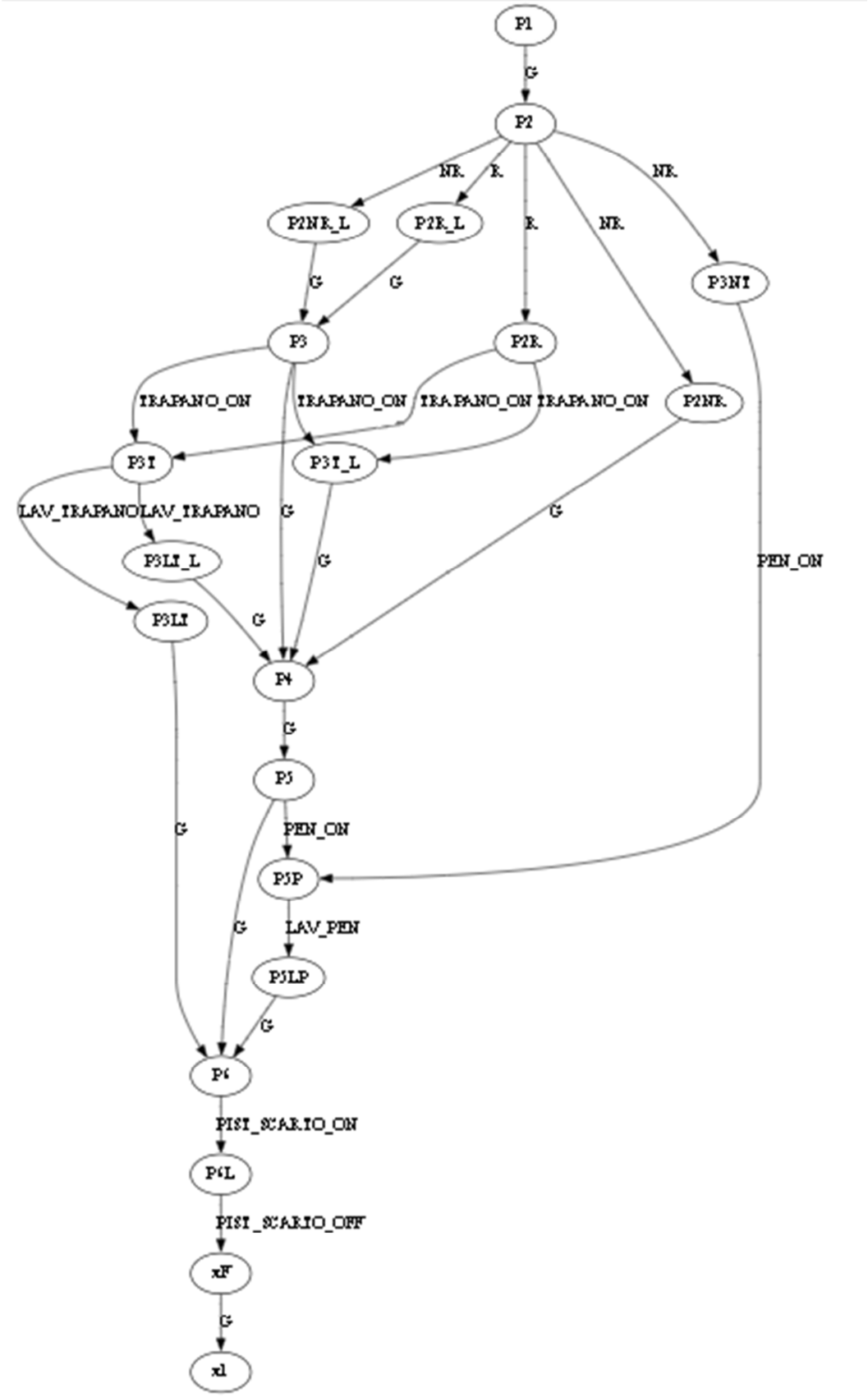

Figura 17. Automa che descrive il funzionamento della giostra senza Fault

Il passo successivo è la costruzione degli automi G^a_{N1} e G^a_{N2}, utilizzando l'algoritmo 1, mostrati rispettivamente nelle Figura 19 e Figura 18, aggiungendo gli stati difettosi *F1, F2 e F2_1* agli automi G_{N1} e G_{N2}.

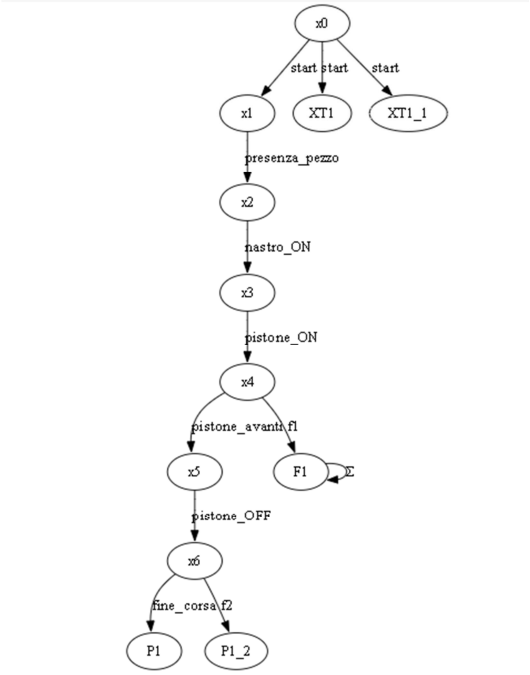

Figura 19. Automa Gn1

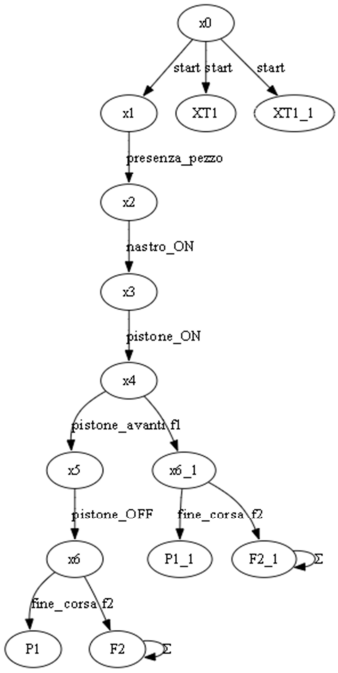

Figura 18. Automa Gn2

Il passo finale dell'Algoritmo 1 è il calcolo dell'automa $\mathrm{G\_c} = \mathrm{G\_N1\hat{}a} \,||\, \mathrm{G}_{\mathrm{N2}}^{\mathrm{a}}$. Successivamente calcoliamo il Petri Net Diagnoser N_D per G_C. Seguendo l'algoritmo 2, otteniamo la rete di Petri a stati, N_C, mostrato in Figura 20.

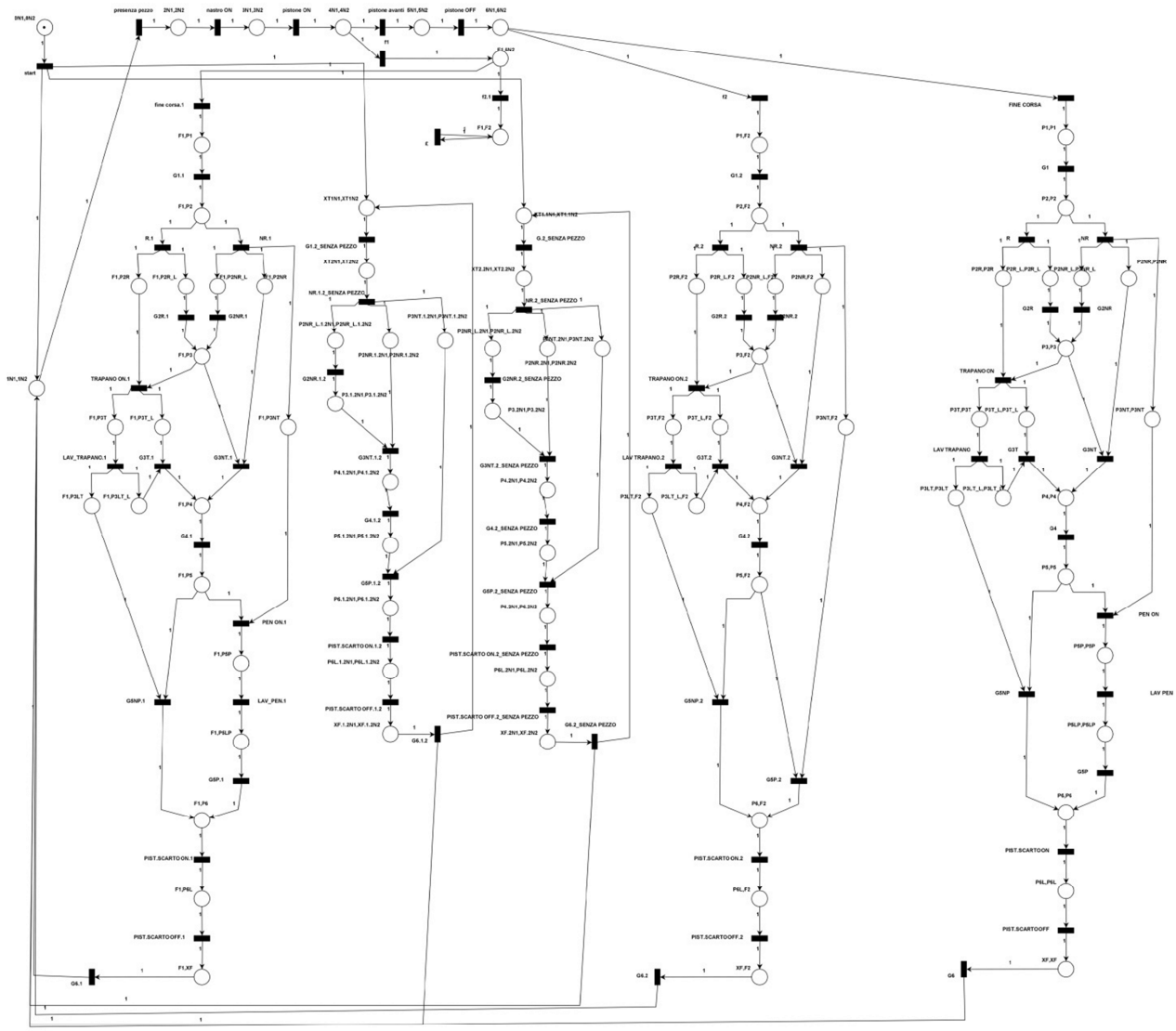

Figura 20a. State Machine Petri Net Nc

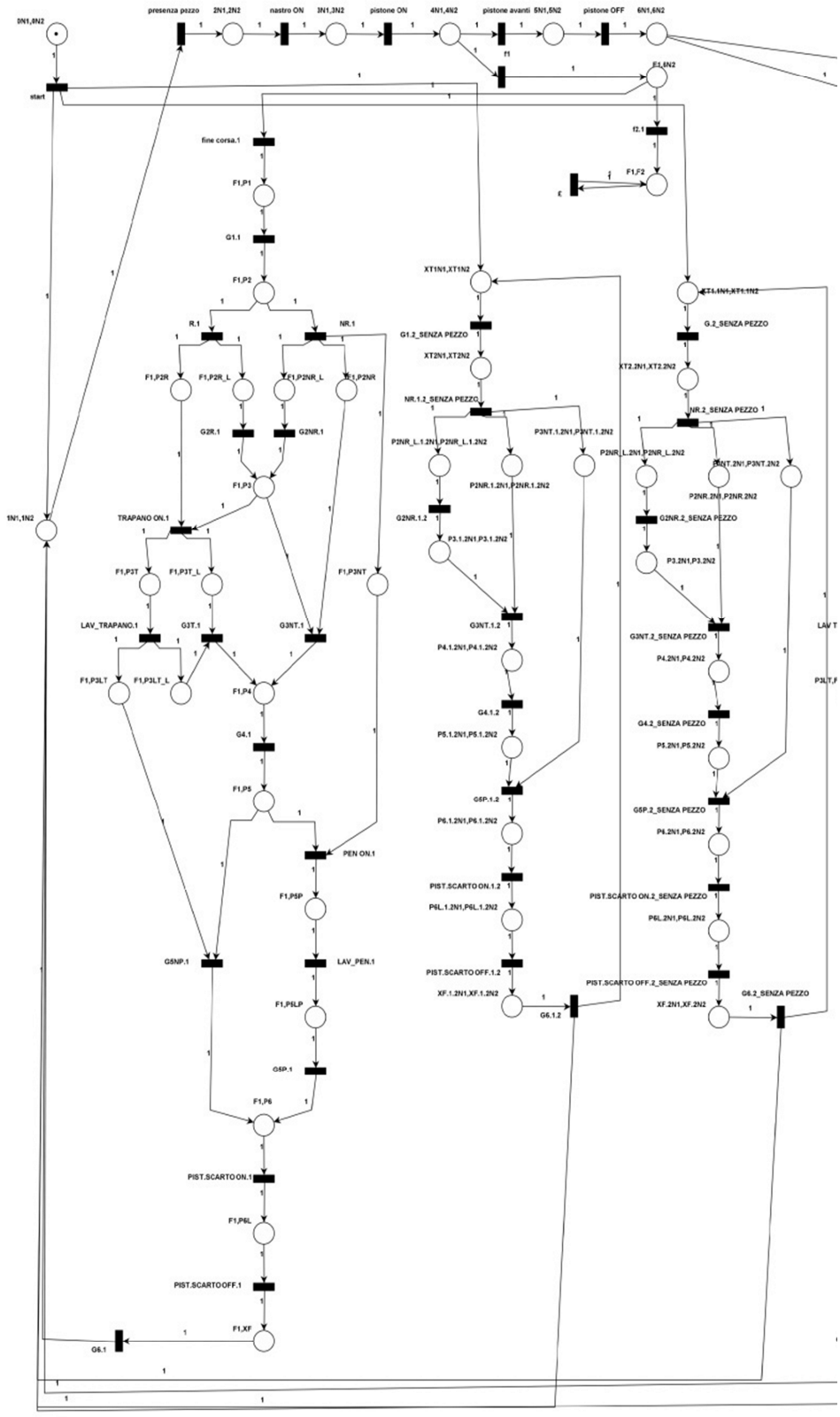

Figura 21b. State Machine Petri Net Nc – ingrandimento parte destra fig. 20a

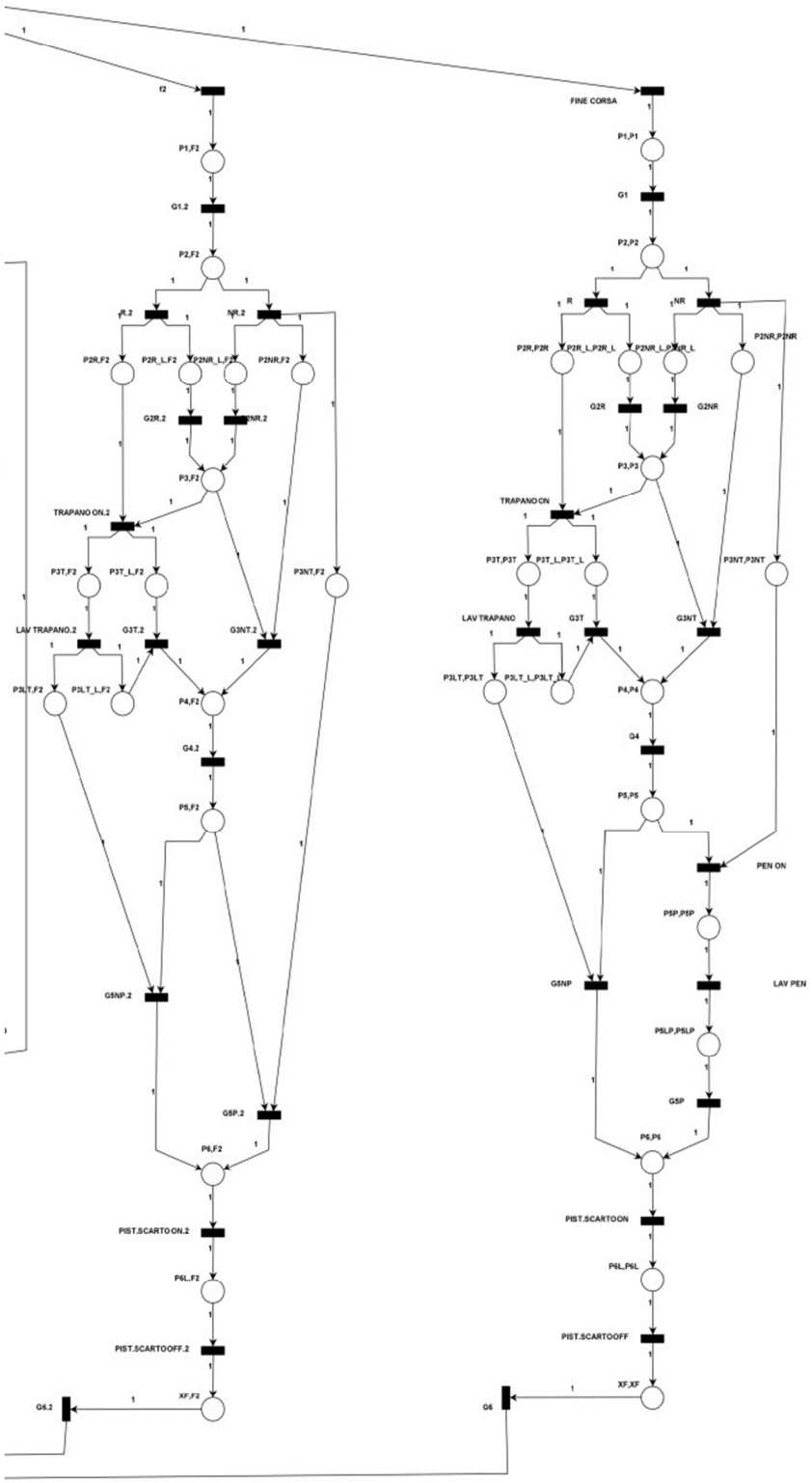

Figura 22c. State Machine Petri Net Nc – ingrandimento parte sinistra fig. 20a

In seguito, dopo l'algoritmo 4, si ottiene la rete di Petri N_{Co}. Dopodiché, utilizzando l'algoritmo 5 viene calcolato il Petri Net State Observer N_{SO}, mostrato in Figura 23.

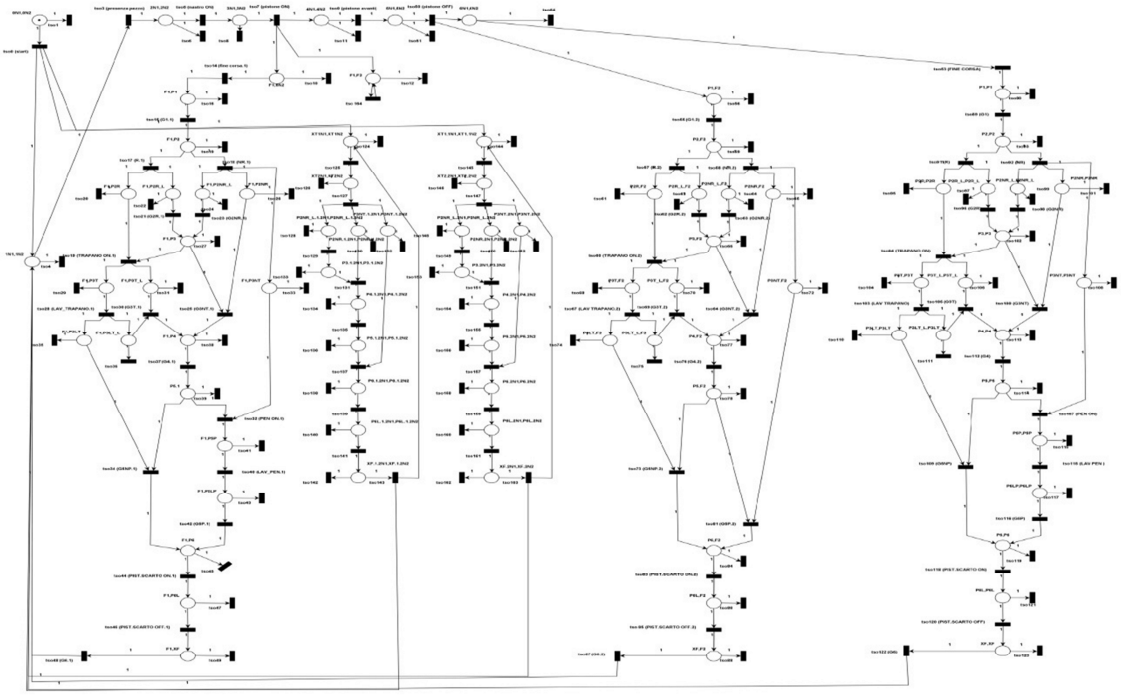

Figura 23. Petri Net State Observer Nso

Infine, seguendo l'algoritmo 6, viene calcolato il Petri Net Diagnoser N_D, presentato in **Errore. L'origine riferimento non è stata trovata.**.

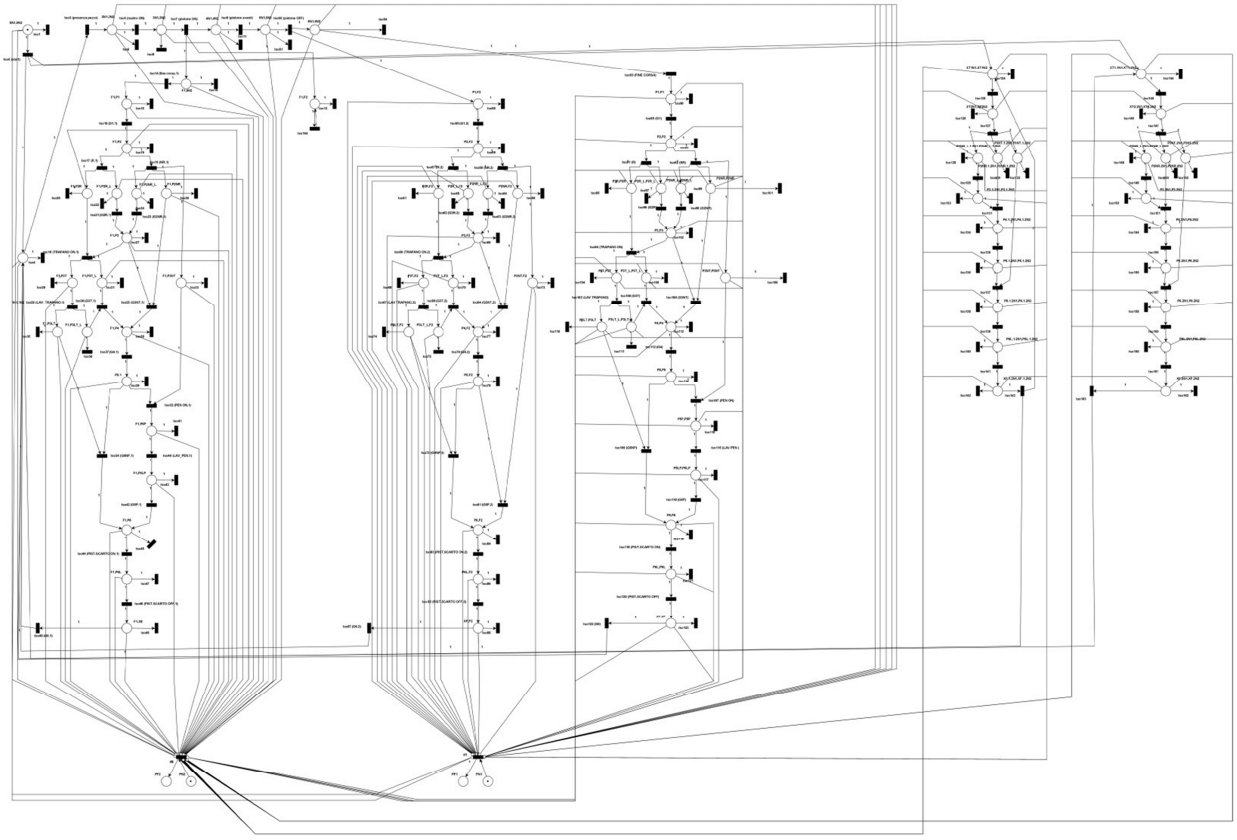

Figura 24a. Petri Net Diagnoser Nd

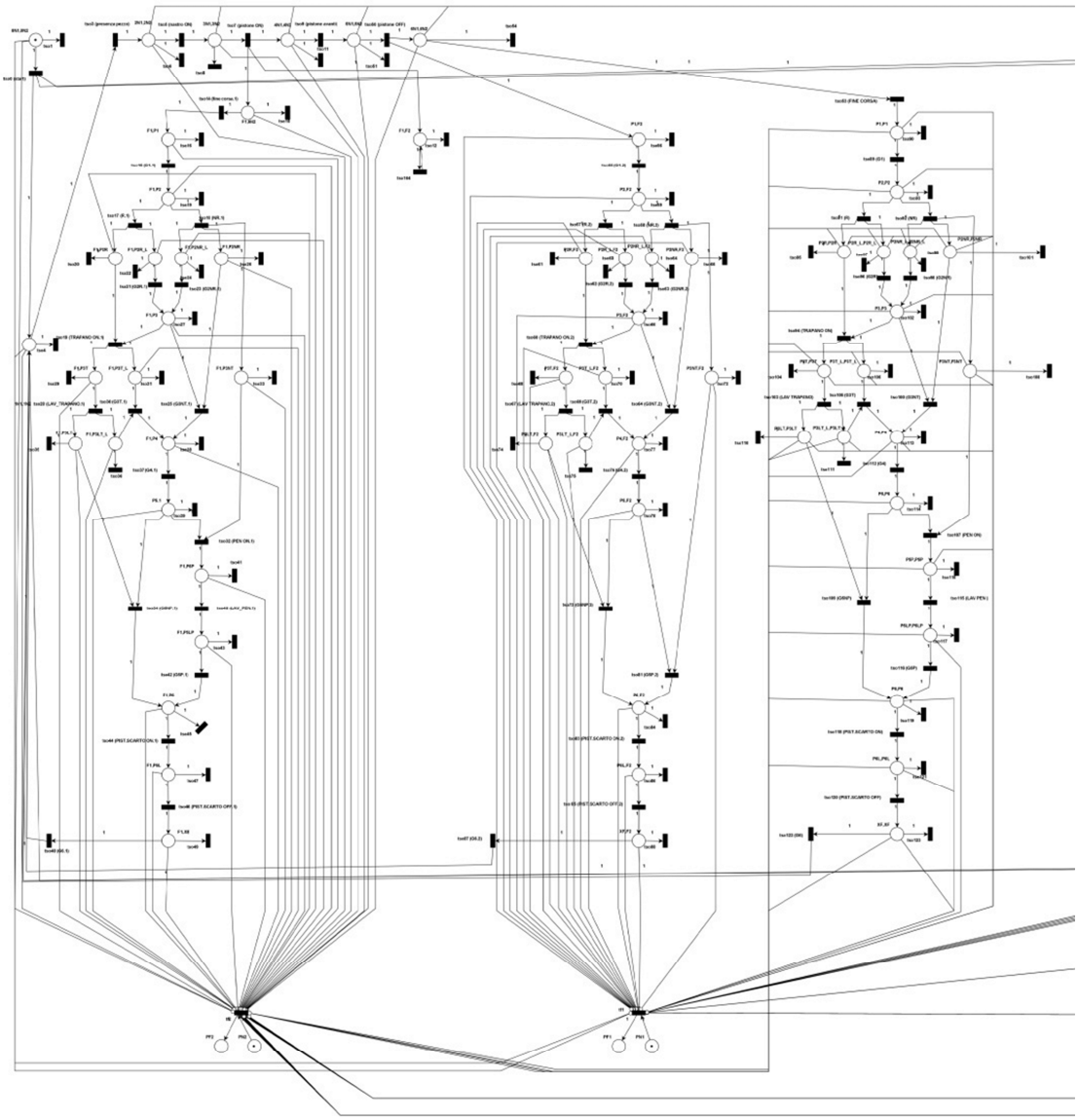

Figura 25b. Petri Net Diagnoser Nd - ingrandimento parte destra fig. 22a

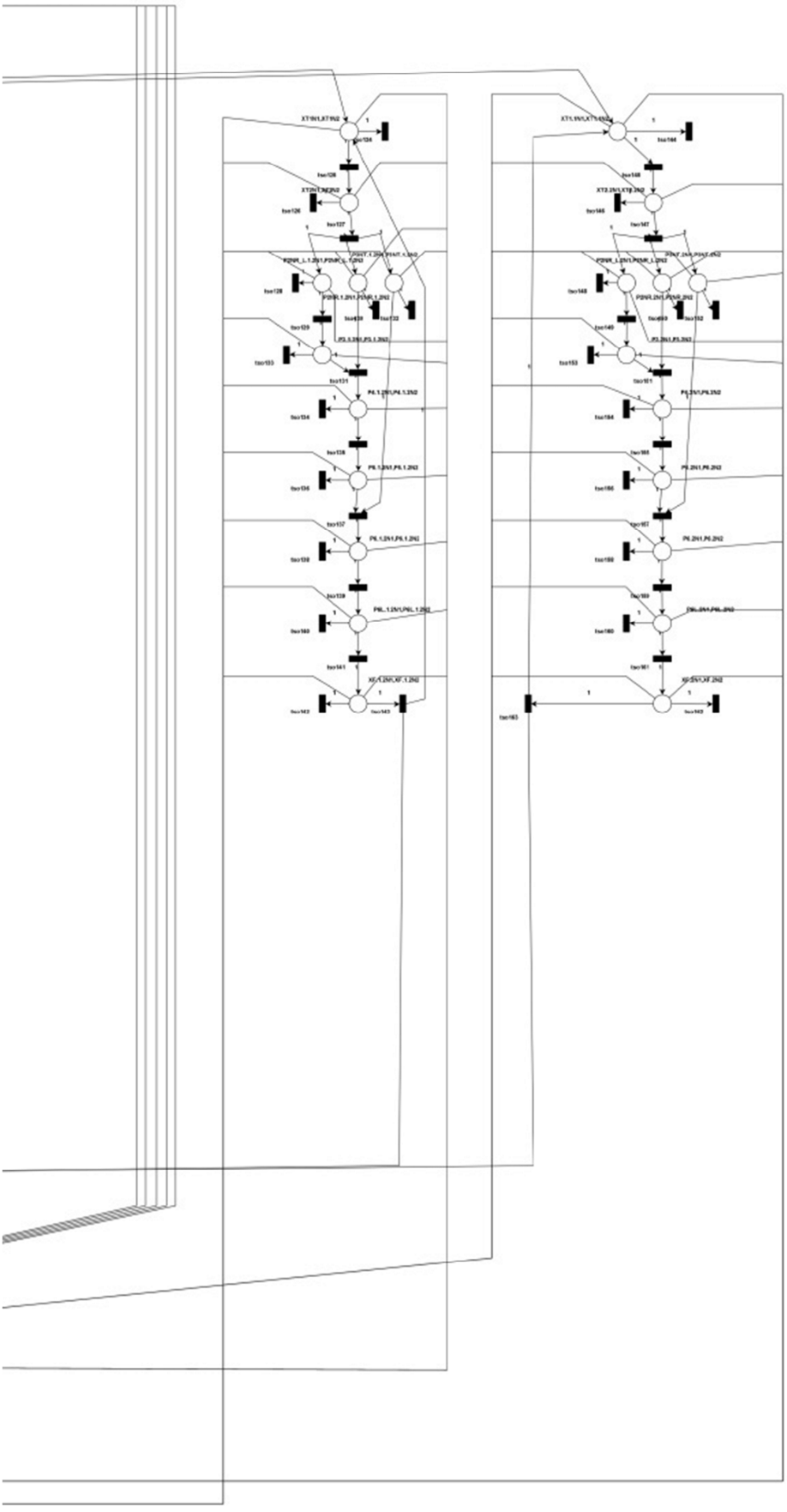

Figura 26c. Petri Net Diagnoser Nd - ingrandimento parte sinistra fig. 22a

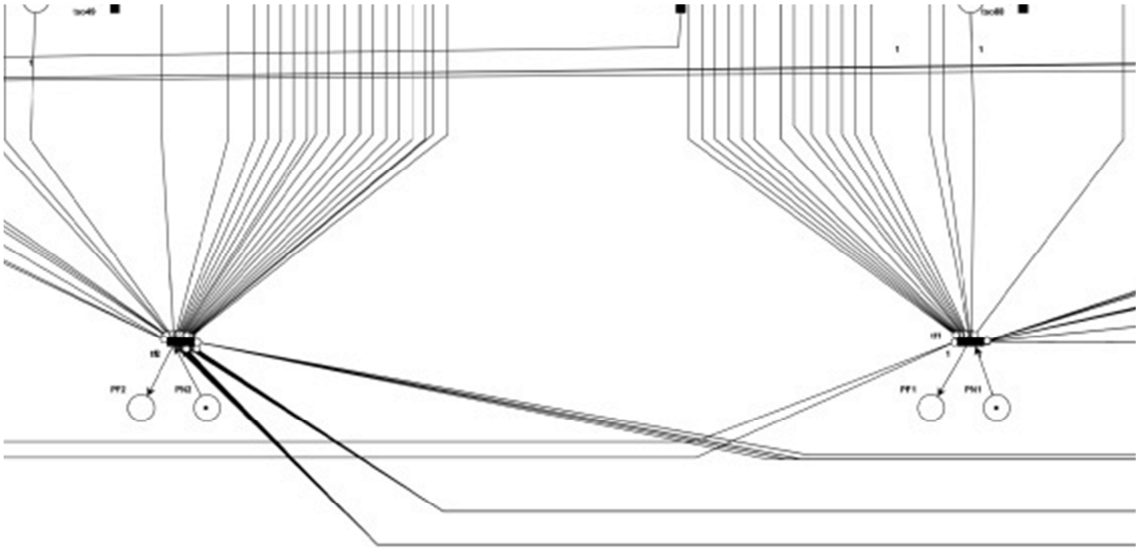

Figura 27d. Petri Net Diagnoser Nd - ingrandimento posti di riconoscimento Fault

Mostriamo ora come viene eseguita la diagnosi online utilizzando N_D facendo due esempi che vanno a diagnosticare il verificarsi dei due eventi di Fault.

Esempio 1. (Diagnosi Fault f_1)

Supponiamo che la traccia difettosa

$s = start\ presenza\ pezzo\ nastroOn\ pistoneON\ f1\ fine\ corsa.1$

sia stata eseguita dal sistema. Quindi, la traccia osservata è

$v = P_o(s) = start\ presenza\ pezzo\ nastroOn\ pistoneON\ fine\ corsa.1.$

Poiché lo stato iniziale di N_D ha un token solo nel posto $(0N_1,0N_2)$, associato allo stato iniziale di G_C , si ha che dopo il verificarsi del primo evento *start*, la transizione t_{SO0} si attiva e l'insieme di posti associati ai possibili stati di G_C che hanno un token è dato da $\{(1N_1, 1N_2), (XT1N_1, XT1N_2), (XT1.1N_1, XT1.1N_2)\}$.

Quindi, quando si osserva il secondo evento *presenza pezzo*, le transizioni t_{SO3} , t_{SO124}, t_{SO144} scattano e l'insieme dei posti con un token è dato da $\{(2N_1, 2N_2)\}$. Si noti che le transizioni t_{SO124} *e* t_{SO144} sono state create in base al passaggio 1 dell'algoritmo 5 per rimuovere il token dei posti che non sono associati a un possibile stato di G_C. Dopo il verificarsi del terzo evento *nastroON*, la transizione t_{SO5} scatta e l'insieme dei posti con un token è dato da $\{(3N_1, 3N_2)\}$. Successivamente si verifica l'evento *pistoneON*, le transizioni t_{SO7} scatta e l'insieme dei posti con un token è dato da $\{(4N_1, 4N_2), (F_1, 6N_2),$

(F1, F2)}. Infine, si verifica l'evento *fine corsa.1*, le transizioni $t_{SO14,}$ t_{SO11} e t_{SO12} e l'unico posto di N_D che rimane con un token è *(F1, P1N2)*. Poiché tutti i posti associati a uno stato di G_C con una coordinata *(q,N1)* non hanno token, la transizione t_{f1} vviene abilitata e si attiva, rimuovendo il token dal posto p_{N1} e aggiungendo un token a p_{F1}, indicando il verificarsi dell'evento di guasto f_1.

Esempio 2. (Diagnosi Fault f_2)

Supponiamo che la traccia difettosa

$s = start\ presenza\ pezzo\ nastroOn\ pistoneON\ pistone\ avanti\ pistoneOff\ f_2\ g$

sia stata eseguita dal sistema. Quindi, la traccia osservata è

$v = P_o(s) =$

$start\ presenza\ pezzo\ nastroOn\ pistoneONpistone\ avanti\ pistoneOffg$.

Poiché lo stato iniziale di N_D ha un token solo nel posto *(0N1,0N2)*, associato allo stato iniziale di G_C , si ha che dopo il verificarsi del primo evento *start*, la transizione t_{SO0} si attiva e l'insieme di posti associati ai possibili stati di G_C che hanno un token è dato da *{(1N1, 1N2), (XT1N1, XT1N2), (XT1.1N1, XT1.1N2)}*.

Quindi, quando si osserva il secondo evento *presenza pezzo*, le transizioni t_{SO3} , $t_{SO124,}$ t_{SO144} scattano e l'insieme dei posti con un token è dato da *{(2N1, 2N2)}*. Dopo il verificarsi del terzo evento *nastroON*, la transizione t_{SO5} scatta e l'insieme dei posti con un token è dato da *{(3N1, 3N2)}*. Successivamente si verifica l'evento *pistoneON*, la transizione t_{SO7} scatta e l'insieme dei posti con un token è dato da *{(4N1, 4N2), (F1, 6N2), (F1, F2)}*. Dopodiché si verifica l'evento *pistone avanti*, le transizioni t_{SO9}, t_{SO10} e t_{SO12} scattano e l'insieme dei posti con un token è dato da *{(5N1, 5N2)}*. In seguito, si verifica l'evento *pistoneOff*, la transizione t_{SO50} scatta e l'insieme dei posti con un token è dato da *{(6N1, 6N2)}*. Infine, si verifica l'evento *g*, le transizioni t_{SO54} e t_{SO55} e l'unico posto di N_D che rimane con un token è *(P2N1, F2)*. Poiché tutti i posti associati a uno stato di G_C con una coordinata *(q,N2)* non hanno token, la transizione t_{f2} viene abilitata e si attiva, rimuovendo il token dal posto p_{N2} e aggiungendo un token a p_{F2}, indicando il verificarsi dell'evento di guasto f_2.

Esempio 3. (Diagnosi nel caso non viene rilevato nessun pezzo)

Supponiamo che la traccia $s = start\ g\ nr$ sia stata eseguita dal sistema. Quindi, la traccia osservata è $v = P_o(s) = start\ g\ nr$.
Poiché lo stato iniziale di N_D ha un token solo nel posto $(0N_1,0N_2)$, associato allo stato iniziale di G_C , si ha che dopo il verificarsi del primo evento *start*, la transizione t_{SO0} si attiva e l'insieme di posti associati ai possibili stati di G_C che hanno un token è dato da $\{(1N_1, 1N_2), (XT1N_1, XT1N_2), (XT1.1N_1, XT1.1N_2)\}$.
Quindi, quando si osserva il secondo evento *g*, le transizioni t_{SO4} , t_{SO125}, t_{SO144} scattano e l'insieme dei posti con un token è dato da $\{(XT2N_1, XT2N_2)\}$. Dopo il verificarsi del terzo evento *nr*, la transizione t_{SO127} scatta e l'insieme dei posti con un token è dato da $\{(P2NR\_L.1.2N_1, P2NR\_L.1.2N_2), (P2NR.1.2N_1, P2NR.1.2N_2), (P2NT.1.2N_1, P2NT.1.2N_2)\}$.
Poiché almeno un posto associato a uno stato di G_C con una coordinata (q,N_1) o (q,N_1) ha un token, allora l'evento di guasto f_2 non si verificato.

Quindi, nel nostro esempio possiamo notare come gli eventi di Fault f_1 e f_2 vengano diagnosticati solo nel caso in cui è presente un pezzo all'interno del sistema.

8. Conclusione

In questa tesi abbiamo approfondito il concetto di diagnosticabiltà dei sistemi ad eventi discreti e analizzato alcune metodologie di diagnosi. Abbiamo fornito una procedura di costruzione del Petri Net Diagnoser (PND) che può essere utilizzato per il rilevamento online dei guasti dei sistemi. Questa procedura richiede meno memoria rispetto agli altri metodi analizzati.
Inoltre, questo lavoro getta le basi per future implementazioni e applicazioni. Una possibile applicazione potrebbe essere quella della conversione del Petri Net Diagnoser in linguaggio SFC (Sequential Function Chart) e in linguaggio Ladder per l'implementazione su PLC (Programmable Logic Control). Poiché il PND è una rete binaria, la sua conversione è abbastanza immediata. Il metodo di conversione è proposto in diversi lavori presenti in letteratura, i quali affrontano anche eventuali problemi di implementazione.

Bibliografia

[1] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen and D. Teneketzis, "Diagnosability of discrete-event systems," in IEEE Transactions on Automatic Control, vol. 40, no. 9, pp. 1555-1575, Sept. 1995, doi: 10.1109/9.412626.

[2] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen and D. Teneketzis, "Failure diagnosis using discrete event models," Proceedings of 1994 33rd IEEE Conference on Decision and Control, Lake Buena Vista, FL, USA, 1994, pp. 3110-3116 vol.3, doi: 10.1109/CDC.1994.411307.

[3] F. G. Cabral and M. V. Moreira, "Synchronous Diagnosis of Discrete-Event Systems," in IEEE Transactions on Automation Science and Engineering, vol. 17, no. 2, pp. 921-932, April 2020, doi: 10.1109/TASE.2019.2951627.

[4] I. Tahiri, A. Philippot, V. Carré-Ménétrier and A. Tajer, "Timed Synthesis Control Approach for Tolerant-Fault Control of Discrete Event Systems (DES)," 2018 International Conference on Control, Automation and Diagnosis (ICCAD), Marrakech, Morocco, 2018, pp. 1-6, doi: 10.1109/CADIAG.2018.8751490.

[5] A. Philippot, M. Sayed-Mouchaweh, V. Carre-Menetrier and B. Riera, "Decentralized Approach to Diagnose Manufacturing Systems," The Proceedings of the Multiconference on "Computational Engineering in Systems Applications", Beijing, China, 2006, pp. 912-918, doi: 10.1109/CESA.2006.4281781.

[6] F. G. Cabral, M. V. Moreira, O. Diene and J. C. Basilio, "A Petri Net Diagnoser for Discrete Event Systems Modeled by Finite State Automata," in IEEE Transactions on Automatic Control, vol. 60, no. 1, pp. 59-71, Jan. 2015, doi: 10.1109/TAC.2014.2332238.

[7] C. G. Cassandras, S. Lafortune. “Introduction to discrete event systems”, Springer Cham New York, 2021, https://doi.org/10.1007/978-3-030-72274-6