Tesi di Laurea Specialistica in Ingegneria Elettronica
Dipartimento di Ingegneria Elettrica ed Elettronica
Università degli Studi di Cagliari

Validazione Piattaforma Software DISC e confronto tra diversi approcci di diagnosi

Laura Marcias

Relatori: Alessandro Giua
Maria Paola Cabasino

ANNO ACCADEMICO 2010-2011

Tesi di Laurea Specialistica in Ingegneria Elettronica
Dipartimento di Ingegneria Elettrica ed Elettronica
Università degli Studi di Cagliari

Validazione Piattaforma Software DISC e confronto tra diversi approcci di diagnosi

Laura Marcias

Relatori: Alessandro Giua
Maria Paola Cabasino

ANNO ACCADEMICO 2010-2011

Ad Andrea e al nostro futuro insieme

Ringraziamenti

Quando si raggiunge un obiettivo così importante non si è mai da soli, senza l'appoggio di chi ci vuole bene nulla sarebbe stato possibile, per questo motivo sento la necessità di ringraziare chi ha contribuito a formare sia la mia preparazione sia la mia personalità.

Il ringraziamento più grande va a mamma e papà per avermi aiutata a realizzare il mio più grande sogno, dal punto di vista economico naturalmente, ma soprattutto per l'appoggio morale che sono sempre stati in grado di darmi, prendendosi cura di me in ogni momento, e sono sicura continueranno a fare.
Un grazie ai miei fratelli Cristian, Michela e Andrea per avermi sopportato, confortato nei momenti difficili e aver sempre fatto il tifo per me, e naturalmente grazie anche ai miei cognati che ormai considero come fratelli, Niamh, Alessandro, Valentina e Michele, che hanno sempre dimostrato di interessarsi a me e ai miei studi pur non capendo nulla di quello che facevo...

Un grazie particolare va ai miei due angeli, Anastasia e Chiara, che per il solo fatto di esistere mi danno felicità e serenità!

Un grazie dal profondo del mio cuore al mio fidanzato Andrea, che con il suo amore, la sua pazienza, il suo buonumore, le sue parole, è riuscito in ogni momento universitario e non solo a spronarmi ad andare avanti, dimostrandosi orgoglioso e soddisfatto dei miei traguardi e perché ha deciso di accompagnarmi anche nel futuro. In fondo questa laurea è anche un po' sua!

Grazie a Signora Gisella e Signor Luciano per avermi accolta a casa loro come una figlia. Grazie a Signora Gilda, ma soprattutto a nonno Francesco per aver "studiato" insieme a me dal primo all'ultimo esame e per tutti gli insegnamenti che mi ha dato, li custodirò con cura nella mia mente.

Un grazie a tutti i miei parenti (sia quelli presenti sia quelli che purtroppo sono volati in cielo, ma io so che sono con me lo stesso), amici e colleghi (nessuno escluso), che mi sono stati vicini con affetto sincero, condividendo le tappe di questa strada e comprendendo (o almeno lo spero) il grande valore che per me ha avuto questa impresa.
Un grazie particolare al mio amico Mauretto per tutti gli angeli messi a disposizione durante la carriera universitaria, ne ha dovuto cercare parecchi per aiutarmi, soprattutto per certi esami e non ha mai fallito nella scelta!
Un grazie particolare alla mia amica Riti per aver condiviso con me interminabili ore di lezione e di studio, ma per fortuna anche delle belle giornate insieme spensierate, dei pranzi come solo noi sappiamo fare, e per il suo fondamentale contributo alla risoluzione dei problemi incontrati durante lo svolgimento della tesi, non so come avrei fatto senza di lei...
Un grazie particolare alla mia amica Giuli per la sincerità del suo affetto e per le cose che ha condiviso con me dentro e fuori l'università, grazie per avermi fatto compagnia durante tutto il percorso!

Un ringraziamento particolare va al Professor Alessandro Giua per gli insegnamenti che mi ha fornito in questi anni di studio e per l'aiuto ricevuto durante la realizzazione di questa tesi.
Un grazie grandissimo alla Dottoressa Maria Paola Cabasino che mi ha guidata e incoraggiata per tutto lo svolgimento della tesi, dimostrandosi sempre competente, disponibile e paziente. E' difficile incontrare persone come lei!
Grazie all'ingegner Luca Contini e agli altri colleghi di Akhela srl per la disponibilità mostrata nell'aiutarmi a risolvere i problemi incontrati.

Infine, un grazie ed un augurio a me stessa...

Indice

Capitolo 1

Introduzione

Sommario

In questa tesi è stata studiata la *Piattaforma Software DISC*, sviluppata da un consorzio di partner europei guidato dall'Università di Cagliari, che permette l'integrazione di strumenti software per l'analisi e il controllo di *Sistemi ad Eventi Discreti* (SED). In particolare, la piattaforma considera come principali modelli di riferimento le reti di Petri e gli automi.

1.1 Progetto DISC

Il progetto DISC è iniziato nel Settembre 2008 il suo termine è previsto entro l'anno 2011. La *Piattaforma Software* ha lo scopo di integrare, insieme ad altri tool già esistenti, gli algoritmi sviluppati durante il corso del progetto. Fondamentalmente si propone due obiettivi: fornire uno strumento che faciliti il trasferimento di queste tecniche agli utenti finali e permettere all'utente di confrontare le diverse metodologie e i diversi tool. In particolare in questa tesi di laurea la *Piattaforma Software* viene utilizzata per mettere a confronto i diversi approcci di diagnosi, ovvero verificare quale procedura risulta migliore per poter determinare se un dato sistema è diagnosticabile oppure no. Prima di poterla utilizzare per i nostri scopi è stato necessario studiare le funzionalità della piattaforma e testarla. I test svolti hanno riguardato principalmente la verifica della funzionalità della piattaforma di convertire i vari formati di file che essa supporta. Tali test hanno messo in risalto vari bachi dei *plugin* e degli *adapter*. Inoltre, si è lavorato in collaborazione con gli sviluppatori di tali convertitori fornendo tutto il sup-

porto necessario per far in modo che i bachi venissero eliminati. Il processo di test, correzione dei convertitori e verifica è stato ripetuto varie volte con esempi random. La piattaforma integra strumenti dedicati sia alla simulazione, che al controllo, che alla diagnosi. Le procedure di conversione tra modelli prescindono dalla funzionalità attesa. In questa tesi, in particolare, sono stati studiati e confrontati alcuni approcci di diagnosi, perciò la piattaforma è stata di fondamentale importanza per la conversione del modello rete di Petri nel modello automa.

1.2 Diagnosi dei SED

E' stato fondamentale studiare il problema della diagnosi dei *sistemi ad eventi discreti*, ossia sistemi dinamici il cui comportamento è governato dal manifestarsi di eventi fisici che causano cambiamenti repentini nello stato del sistema. Gli stati dei SED riflettono lo stato normale e quello di guasto dei componenti del sistema, mentre gli eventi di guasto fanno parte dell'insieme degli eventi. Il problema è scoprire il verificarsi di tali eventi. Il concetto chiave alla base della diagnosi e della diagnosticabilità è quello di guasto, che viene inteso come qualsiasi scostamento del sistema dal suo comportamento normale o comunque previsto. La diagnosi è il processo che individua le anormalità nel comportamento del sistema e isola la causa o la sorgente di questa anormalità.

La diagnosticabilità è il processo che consiste nel determinare a priori se un sistema è diagnosticabile o no, ossia se sia possibile ricostruire l'occorrenza di eventi di guasto osservando parole di lunghezza finita. Partendo da uno stesso sistema si è voluto mettere a confronto lo studio della diagnosticabilità usando come modello le reti di Petri e gli automi. Il confronto ha come obiettivo quello di stabilire quale tra i diversi approcci di diagnosi sia il più efficiente dal punto di vista della cardinalità dello spazio di stato ma anche dai tempi impiegati a stabilire la diagnosticabilità di un dato sistema. Per quanto riguarda le reti di Petri si è fatto riferimento ai toolbox realizzati da Pocci [2] e Perria [4].

Il primo si basa sulla metodologia trattata da Cabasino *et al.* in [13], in cui la diagnosticabilità viene valutata tramite l'analisi di due grafi il *Modified Basis Reachability Graph* (MBRG) e il *Basis Reachability Diagnoser* (BRD). Tale toolbox è stato inserito all'interno della piattaforma con il nome *PN_DIAG*.

Il secondo, invece, inserito come *PN_DIAG_2* all'interno della piattaforma, valuta la diagnosticabilità attraverso il *Modified Basis Reachability Graph* (MBRG) e il *Modified Basis Reachability Diagnoser* (MBRD).

Per quanto riguarda gli automi, si è utilizzata la libreria UMDES [24], sviluppata per modellare gli automi a stati finiti e per la verifica di diverse proprietà tra cui la diagnosticabilità. Tale libreria è stata sviluppata dal gruppo DES (*Discrete Event System*) dell'Università del Michigan sotto la coordinazione dei Professori Lafortune e Teneketzis [30].

L'analisi è stata condotta sulla cardinalità dello spazio di stato del sistema, del suo diagnosticatore e dei tempi impiegati per calcolarlo. Ci si aspetta che il toolbox realizzato da Perria sia il più efficiente, ma per poterlo dire, dobbiamo avere prove di ciò: a questo vuole rispondere la tesi.

1.3 Stato dell'arte

Il problema della diagnosi dei guasti ha ricevuto una grande attenzione nella letteratura. Diversi e originali sono gli approcci teorici mediante automi, proposti in tal senso, si veda ad esempio [11], [17], [21], [25], [29], [37].

Recentemente il problema della diagnosi è stato affrontato anche usando una particolare classe di SED, ossia le reti di Petri. L'uso delle reti di Petri offre dei significativi vantaggi per la loro doppia rappresentazione: grafica e matematica. Tra i vari contributi apportati in questo settore possiamo citare i lavori di Ushio *et al.* [35], Aghasaryan *et al.* [7], Benveniste *et al.* [9], Jiroveanu e Boel [10], [22], Basile *et al.* [8], Genc e Lafortune [18]. La maggior parte di questi approcci richiede una completa enumerazione dello spazio di stato.

In questa tesi si farà riferimento alla teoria sviluppata da Cabasino *et al.* [14], [13] relativamente alla diagnosi dei SED. La differenza fondamentale tra gli approcci precedentemente citati, e l'approccio alla diagnosi usando reti di Petri in [14], [19] sta nel concetto di *marcatura di base*. L'approccio tramite marcatura di base permette di enumerare solo un sottoinsieme dello spazio di stato. Per quanto riguarda la diagnosticabilità, il suo studio è stato ampiamente affrontato nell'ambito degli automi; mentre sono stati proposti, pochi risultati relativamente alle reti di Petri (vedi [12], [16], [30], [31], [35], [36]). In questa tesi faremo riferimento al lavoro di Cabasino *et al.* ([13]).

1.4 Contributi della tesi

I contributi apportati da questa tesi sono sostanzialmente i seguenti:

- testare la release della piattaforma, scaricabile da [1], dove sono indicati i requisiti che il sistema deve avere per poter supportare la piattaforma. Nel momento dell'installazione è opportuno essere collegati alla rete internet, perchè Windows Installer 3.1 e .Net framework 3.0 verranno automaticamemte scaricati ed installati se non sono già presenti;

- testare la correttezza dei convertitori (*plugin* e *adapter*) appartenenti alla piattaforma; il loro compito è quello di convertire un dato file nel formato di interesse per l'analisi;

- individuare le cause degli errori presenti nei *plugin* e negli *adapter* mal funzionanti e informare gli sviluppatori dei convertitori di quali correzioni necessitano;

- testare la funzionalità della piattaforma nel suo complesso; grazie all'uso degli script è possibile sottoporre la piattaforma a diverse combinazioni di conversione;

- confrontare i diversi approcci di diagnosi. I risultati mostrano che al momento il migliore è quello su cui è basato il software *PN_DIAG_2*. Infatti, esso permette di avere una risposta sulla diagnosticabilità di un dato sistema in tempi accettabili e soprattutto tramite grafi (MBRG e MBRD) la cui cardinalità è inferiore a quella che si avrebbe nel caso si considerasse l'approccio basato sugli automi.

1.5 Struttura della tesi

La struttura della tesi è la seguente.

Il Capitolo 2 contiene una breve descrizione sulla diagnosi in generale e sulla diagnosi dei SED in particolare. Inoltre, viene richiamato il formalismo degli *automi finiti deterministici* (AFD), degli *automi finiti non deterministici* (AFN) e delle *reti di Petri* (RdP), in particolare delle *reti posto/transizione* (P/T), modello che non permette di rappresentare la temporizzazione degli eventi ma solo l'ordine con cui questi si verificano. Vengono richiamati alcuni concetti di base tra cui: matrici Pre e $Post$, marcatura, abilitazione e scatto, linguaggio di una rete e vengono poi definite alcune importanti proprietà relative alle reti di Petri. Si è poi spiegato in cosa consistono i grafi generati dai toolbox *PN_DIAG* e *PN_DIAG_2* per la verifica della diagnosticabilità di un dato sistema.

Il Capitolo 3 contiene la descrizione dettagliata della *Piattaforma Software*. Alla fine del progetto la *Piattaforma Software* verrà distribuita alle università e alle industrie tramite il *web-server* del progetto [1].

Nel Capitolo 4 verranno richiamati brevemente i software testati all'interno della tesi. In particolare, daremo una descrizione del software *PN_DIAG* di Pocci [28] e il software *PN_DIAG_2* di Perria [26], entrambi riguardanti il problema della diagnosticabilità delle reti di Petri. Inoltre verrà richiamato il software UMDES (*Discrete Event Systems Modeled By Finite State Machine*). Quest'ultimo software inserito all'interno del tool *Desuma*, contiene comandi per la manipolazione di una macchina a stati finiti, per la diagnosi dei guasti e per il controllo supervisivo. La parte interessante per questo lavoro è quella riguardante la diagnosi dei guasti. Tutti questi software sono stati testati sia all'interno che all'esterno della *Piattaforma Software*.

Il Capitolo 5 illustra i problemi riscontrati durante la fase di test della piattforma. In particolare, sono stati trovati dei bachi all'interno dei *plugin* e degli *adapter* che impedivano il funzionamento corretto dei toolbox (*PN_DIAG* e *PN_DIAG_2*) inseriti all'interno della *Piattaforma Software*. Inoltre, poichè lo scopo della tesi è confrontare i risultati forniti dai software *PN_DIAG* e *PN_DIAG_2* con quelli forniti da *Desuma*, è stato testato anche quest'ultimo toolbox, ed è stato trovato un baco all'interno del file eseguibile *dcycle*. Tale baco è stato comunicato tempestivamente al Professor Lafortune.

Nel Capitolo 6 riportiamo alcuni esempi di reti di Petri di cui abbiamo studiato la diagnosticabilità utilizzando i tool *PN_DIAG* e *PN_DIAG_2*, e confrontiamo i risultati ottenuti, sempre per quanto riguarda la diagnosticabilità utilizzando i file eseguibili *diag_UR* e *dcycle* appartenenti alla libreria UMDES. Il tutto è stato fatto utilizzando la *Piattaforma Software*. Vengono riportate per chiarezza d'esposizione anche le figure rappresentanti le reti di Petri e gli schemi seguiti nell'utilizzo della piattaforma. Viene inoltre preso in considerazione un modello fisico parametrico, utilizzato per fare un confronto tra i diversi approcci per la verifica della diagnosticabilità. Sono state fatte delle simulazioni per diversi valori dei parametri e raccolti i risultati ottenuti facendo le dovute considerazioni.

Il Capitolo 7 è il capitolo conclusivo della tesi.

Capitolo 2

La diagnosi

Sommario

In questo capitolo verrà introdotto il concetto di diagnosi in riferimento ai *Sistemi ad Eventi Discreti* (SED). Inoltre, verrà richiamato il formalismo degli *Automi Finiti Deterministici* (AFD), degli *Automi Finiti Non deterministici* (AFN) e delle *Reti di Petri* (RdP), in particolare delle *reti posto/transizione* (P/T). Per maggiori dettagli facciamo riferimento al libro di Di Febbraro e Giua [6].

2.1 La diagnosi

La scoperta e l'isolamento dei guasti nei sistemi industriali è un tema che ha ricevuto moltissima attenzione negli ultimi decenni. Un guasto è definito come uno scostamento del sistema dal suo comportamento normale; la diagnosi è il processo che individua le anormalità nel comportamento del sistema e isola la causa o la sorgente di questa anormalità. I guasti in un sistema industriale potrebbero nascere per diversi motivi come un errore nel progetto, un'apparecchiatura mal funzionante, uno sbaglio dell'operatore, e così via. Ci sono tre principali fattori che spingono lo studio del problema della diagnosi dei guasti:

1. i guasti sono inevitabili;
2. la diagnosi dei guasti è importante;
3. la diagnosi dei guasti è difficile.

I guasti sono inevitabili nei moderni e complessi ambiti industriali. Così come la tecnologia evolve e come noi continuiamo a costruire sistemi sempre più complessi e funzionali, pretendendo da questi sempre maggiori prestazioni, così la complessità di questi sistemi aumenta. Di conseguenza aumenta anche la possibilità che il sistema si guasti, perciò è necessario rendere sicuri i nostri progetti, migliorare le tecniche di controllo e la formazione degli operatori, in caso contrario i guasti saranno inevitabili. Infatti, data la complessa interazione fra i componenti, i sottosistemi e i processi, un guasto del sistema potrebbe essere considerato come un evento normale, o come una caratteristica della maggior parte dei sistemi industriali.

Essendo i guasti inevitabili, si ha il bisogno di un efficace strumento per la loro scoperta, perchè possono provocare conseguenze non solo sui sistemi coinvolti, ma sulla stessa società. Quindi due primi fattori che ci spingono allo studio di questo problema sono la sicurezza e l'affidabilità. Inoltre, metodi efficaci nella diagnosi dei guasti non solo aiutano ad evitare effetti indesiderati, ma possono anche aumentare la qualità e il prestigio delle industrie. La migliore qualità delle prestazioni, l'integrità e l'affidabilità di un prodotto, la riduzione dei costi per la manutenzione delle apparecchiature e dei servizi sono alcuni dei maggiori benefici che possono portare degli schemi efficaci di diagnosi, specialmente per quei prodotti e servizi orientati alle industrie come quelle di controllo di ambienti industriali e domestici, di servizi d'automazione, industrie manifatturiere di automobili e semiconduttori. Quindi metodi efficaci e tempestivi di diagnosi dei guasti possono aumentare la sicurezza, l'affidabilità, la qualità e l'economia dei processi industriali.

La natura complessa dei sistemi industriali non solo aumenta la potenziale avvenuta di un guasto, ma rende i problemi diagnostici molto difficili e impegnativi. Molti sistemi sono dotati di indicatori quali allarmi, luci di avviso, e così via, per indicare lo stato del sistema agli operatori che stanno monitorando il suo comportamento ed aiutarli nel loro ragionamento sulla diagnosi. Però, il costo e la fattibilità tecnologica limitano il numero di sensori e quindi il numero delle variabili di sistema che possono essere direttamente monitorate. Le complesse e spesso non evidenti interazioni ed accoppiamenti fra i componenti del sistema rendono le deduzioni un'attività impegnativa, specialmente se le decisioni devono essere prese in fretta. Ma, anche avendo a disposizione tutti i sensori di cui si ha bisogno, il problema resta comunque difficile. Infatti, risulta complicato confrontare i dati provenienti da un largo numero di sensori, spesso apparentemente contraddittori, specialmente in situazioni di guasto, e correlarli con i possibili guasti. Questo qualche volta porta gli operatori a ignorare gli allarmi. In più, ci sono stati nella storia degli incidenti industriali, causati da valori sbagliati o letti

erroneamente, e/o interpretazini sbagliate degli operatori sulle informazioni dei sensori.

Da quanto detto, emerge che i meccanismi per una tempestiva e accurata diagnosi dei guasti sono veramente essenziali. Tale bisogno è stato ben capito e apprezzato sia nell'industria che nella ricerca. Una grande quantità di ricerche e sforzi sono stati spesi e tuttora vengono dedicati alla creazione e allo sviluppo di un sistema diagnostico automatico e sono stati proposti una varietà di schemi, che differiscono nella struttura teorica e nella filosofia di progetto e implementazione.

I metodi per la diagnosi dei guasti possono essere classificati nel seguente modo:

- metodi basati sull'albero dei guasti;
- metodi basati sui modelli analitici;
- metodi basati sulla costruzione di modelli a partire dal ragionamento;
- metodi basati sui *Sitemi ad Eventi Discreti*.

Da un punto di vista implementativo questi sistemi dignostici possono essere classificati come *off-line* e *on-line*. Nel primo caso il sistema che deve essere diagnosticato non è operativo e può essere pensato come un sistema in un banco di prova. La procedura diagnostica prevede l'esecuzione di una serie di comandi di prova, poi si osservano i risultati di questi e si traggono le conclusioni sull'insieme dei possibili stati in cui il sistema potrebbe trovarsi. Nel secondo caso, il sistema si assume in condizioni di normale funzionamento, e lo scopo è di emettere una sequenza di comandi e identificare lo stato del sistema. Al contrario del caso di diagnosi *off-line*, durante il processo di diagnosi bisogna rendersi conto del possibile verificarsi di eventi non osservabili.

In questa tesi si farà riferimento alla diagnosi basata sui *sistemi ad eventi discreti*.

2.2 La diagnosi dei sistemi ad eventi discreti

I *sistemi ad eventi discreti* sono dei sistemi dinamici il cui comportamento è governato dal manifestarsi di eventi fisici che causano cambiamenti repentini nello stato del sistema. Gli stati del SED riflettono lo stato normale e quello di guasto dei componenti del sistema, mentre gli eventi di guasto fanno parte dell'insieme degli eventi. Il problema è scoprire il verificarsi di tali eventi. Il vantaggio di

questo approccio è che non richiede dettagli approfonditi per costruire il modello del sistema che deve essere diagnosticato e quindi è particolarmente adatto per la diagnosi di sistemi che sono difficili da modellare. Tipici esempi includono estesi e/o complessi sistemi di riscaldamento, ventilazione e aria condizionata, impianti di potenza e processi manifatturieri di semiconduttori e di automobili. Inoltre, tale approccio alla diagnosi è appropriato per guasti notevoli o improvvisi che portano significativi cambiamenti allo stato dei componenti del sistema, ma non necessariamente portano il sistema alla rottura.

I SED si dividono in due grandi famiglie: i SED logici e i SED temporizzati, questi ultimi a loro volta si dividono in SED deterministici e SED stocastici. Nei modelli logici la traccia degli eventi è costituita semplicemente da una sequenza di eventi $\{e_1, e_2, ...\}$, in ordine di occorrenza, senza alcuna informazione circa i tempi di occorrenza degli eventi. Nei modelli temporizzati, invece, la traccia degli eventi è costituita da una sequenza di coppie $\{(e_1, \tau_1), (e_2, \tau_2), ...\}$ dove ogni elemento e_i è accoppiato al suo tempo di accadimento τ_i, eventualmente stocastico. Dato uno stato iniziale x_0, la traiettoria dello stato verrà costruita nel tempo come la sequenza di stati $\{x_0, x_1, x_2, ...\}$ risultanti dall'accadimento della sequenza di eventi, per i modelli logici non è possibile specificare gli istanti di tempo in cui avvengono le transizioni di stato, mentre nei modelli temporizzati si sa che le transizioni di stato avvengono negli stati di occorrenza degli eventi. In questo lavoro di tesi verranno analizzati i SED logici, in particolare verrà effettuata la diagnosi sugli *automi* (per approfondimenti si rimanda a Lewis [20] e Cassandras e Lafortune [15] e sulle *reti di Petri* che traggono origine dal lavoro di un ricercatore tedesco Carl Adam Petri [27].

2.3 Automi e reti di Petri

2.3.1 Introduzione

Un *Sistema ad Eventi Discreti* è un sistema il cui comportamento dinamico è caratterizzato dall'accadimento asincrono di eventi che individuano lo svolgimento di attività di durata non necessariamente nota. Formalmente, un SED è caratterizzato da:

- un insieme E degli eventi accadibili;
- uno spazio di stato costituito da un insieme discreto X;

- un'evoluzione dello stato *event-driven*, cioè regolata dagli eventi: lo stato evolve nel tempo solo in dipendenza dell'accadimento di eventi asincroni, appartenenti all'insieme E.

L'equazione che descrive l'evoluzione dello stato a partire dallo stato iniziale x_0 è

$$x_{k+1} = \delta(x_k, e_k)$$

dove

- x_{k+1} è lo stato del sistema dopo l'accadimento dell'evento k-esimo;
- e_k è il k-esimo evento accaduto dall'istante iniziale considerato, che fa transire lo stato da x_k a x_{k+1};
- $\delta : X \times E \longrightarrow X$ è la funzione di transizione di stato.

2.3.2 Automi Finiti Deterministici

Definizione 2.3.1. *Un Automa Finito Deterministico è una quintupla che si denota $G = (X, E, \delta, x_0, X_m)$ dove:*

- *X è un insieme finito di stati;*
- *E è un insieme finito di eventi (cioè un afabeto di simboli);*
- *$\delta : X \times E \longrightarrow X$ è la funzione di transizione; $\delta(x, e)$ indica lo stato raggiunto dall'automa quando si verifica l'evento e a partire dallo stato x;*
- *$x_0 \in X$ è lo stato iniziale;*
- *$X_m \subseteq X$ è l'insieme di stati finali (o stati marcati).* ■

Un automa può essere descritto da un grafo in cui ogni stato è rappresentato con un nodo, lo stato iniziale da un nodo con una freccia, e un insieme di stati finali da un nodo cerchiato. Se $\bar{x} = \delta(x, e)$ vi sarà un arco orientato dal nodo x al nodo $\bar{x}$ etichettato con il simbolo e per rappresentare la transizione da x a $\bar{x}$; tale arco viene anche indicato con il termine e-transizione.

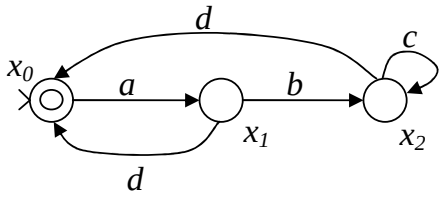

Figura 2.1: Un automa finito deterministico

Esempio 2.3.1. *La Figura 2.1 rappresenta un automa con* $X = \{x_0, x_1, x_2\}$*, alfabeto* $E = \{a, b, c, d\}$*, stato iniziale* x_0 *e insieme di stati finali* $X_m = \{x_0\}$*. La funzione di transizione è data dalla seguente tabella:*

δ	a	b	c	d
x_0	x_1	–	–	–
x_1	–	x_2	–	x_0
x_2	–	–	x_2	x_0

Ad esempio, il simbolo x_1 *all'intersezione fra la riga* x_0 *e la colonna* a *indica che vale* $\delta(x_0, a) = x_1$*. Una casella vuota, come quella all'intersezione fra la riga* x_0 *e la colonna* b*, indica che la corrispondente transizione non è definita. La* e*-transizione che parte dal nodo* x_2 *e arriva allo stesso nodo è detta cappio. L'automa in Figura 2.1, ad esempio, può descrivere in modo semplificato il comportamento di una macchina spenta (stato* x_0*) che su comando di un operatore (evento* a*) viene accesa (stato* x_1*). Una volta accesa viene attrezzata (evento* b*) raggiungendo lo stato di avviamento (stato* x_2*) potendo iniziare una lavorazione su un pezzo (evento* c*). Una volta avviata, la macchina può continuare a lavorare un pezzo dopo l'altro un numero indefinito di volte (l'evento* c *infatti può sempre verificarsi una volta raggiunto lo stato* x_2*). Da un qualunque stato di funzionamento l'operatore può spegnere la macchina (evento* d*). Si considera come finale lo stato* x_0*, per indicare che si desidera riportare la macchina nello stato in cui è spenta al termine di un ciclo di lavorazione.* ■

Occorre notare che poichè in un AFD δ è una funzione parziale, per ogni $x \in X$ e per ogni $e \in E$ il valore della transizione $\delta(x, e)$ può non essere definito o essere univocamente definito. Se non fosse definito, non ci sarà una e-transizione uscente dal nodo x. Se fosse definito, vi sarà una e-transizione uscente dal nodo x. Non è possibile, però, che dallo stesso nodo escano due o più archi etichettati

con lo stesso simbolo. Ancora si osservi che, poichè ad ogni transizione corrisponde un evento, le etichette delle transizioni uscenti da uno stato x indicano quali eventi possono verificarsi in tale stato.

Il comportamento di un automa è dato dalle possibili evoluzioni, cioè sequenze di transizioni, che esso può generare. Ad ogni sequenza di transizioni corrisponde una produzione.

Definizione 2.3.2. *Dato un AFD* $G = (X, E, \delta, x_0, X_m)$ *si definisce produzione una sequenza*

$$x_{j_0} \xrightarrow{e_1} x_{j_1} \xrightarrow{e_2} ...x_{j_{k-1}} \xrightarrow{e_k} x_{j_k}$$

dove per ogni $i = 0, ..., k$ *vale* $x_{j_i} \in X$ *e inoltre da ogni stato* $x_{j_{i-1}}$ *il verificarsi dell'evento* e_i *porta allo stato* x_{j_i}*, cioè per ogni* $i = 1, ..., k$ *vale* $x_{j_i} = \delta(x_{j_{i-1}}, e_i)$*. Si dice anche che tale produzione genera la parola* $w = e_1 e_2 ... e_k$ *partendo dallo stato* x_{j_0} *e raggiungendo lo stato* x_{j_k}*.* ■

Ad esempio una possibile produzione dell'automa di Figura 2.1 è la seguente:

$$x_0 \xrightarrow{a} x_1 \xrightarrow{b} x_2 \xrightarrow{c} x_2 \xrightarrow{c} x_2$$

Questa produzione raggiunge lo stato x_2 a partire dallo stato x_0 e descrive un'evoluzione in cui la macchina viene accesa, attrezzata e lavora due parti. Tale produzione genera la parola $w = abcc$. Si noti che una produzione può essere definita a partire da uno stato qualunque e non necessariamente dallo stato iniziale. Poichè δ è una funzione, è facile capire che non possono esistere due produzioni diverse che partono dallo stesso stato e generano la stessa parola. Poichè a ogni evoluzione è associata una parola formata da simboli dell'alfabeto degli eventi E, all'insieme delle possibili evoluzioni che partono dallo stato iniziale è possibile associare un linguaggio $L \subseteq E^*$.

Definizione 2.3.3. *Dato un AFD* $G = (X, E, \delta, x_0, X_m)$ *si dice che una parola* $w \in E^*$ *è*

- *generata da G se* $\delta(x_0, w)!$*, cioè se esiste una produzione che genera w a partire dallo stato iniziale;*
- *accettata (o marcata) da G se* $\delta^*(x_0, w) = x \in X_m$*, cioè se esiste una produzione che genera w a partire dallo stato iniziale e raggiunge uno stato finale.* ■

Considerando la parola $abcc$, essa è generata dall'automa in Figura 2.1 perchè $\delta^*(x_0, abcc) = x_2$; tuttavia non accettata perchè x_2 non è uno stato finale. Viceversa la parola ad è accettata (e dunque anche generata) perchè $\delta^*(x_0, ad) = x_0$ e x_0 è uno stato finale. Infine la parola ac non è generata (e dunque nemmeno accettata) perchè $\delta^*(x_0, ac)$ non è definita. Da notare che la parola vuota è sempre generata e viene accettata solo se lo stato iniziale è anche finale.

Definizione 2.3.4. *Dato un AFD $G = (X, E, \delta, x_0, X_m)$ è possibile associare ad esso due linguaggi:*

- *Il linguaggio generato*

$$L(G) = \{w \in E^* \mid \delta^*(x_0, w)!\} \subseteq E^*,$$

  cioè il linguaggio che costituisce l'insieme di tutte le parole generate.

- *Il linguaggio accettato (o marcato)*

$$L(G) = \{w \in E^* \mid \delta^*(x_0, w) \in X_m\} \subseteq L(G),$$

  cioè il linguaggio che costituisce l'insieme di tutte le parole accettate. ■

E' importante osservare che il linguaggio generato da un AFD è necessariamente chiuso per prefisso, ovvero se una parola può essere generata allora devono poter essere generati tutti i suoi prefissi, cioè vale $L(G) = \overline{L(G)}$. Viceversa, il linguaggio accettato da un AFD non è necessariamente chiuso per prefisso, perchè non sempre tutti i prefissi di una parola accettata sono accettati. Ad esempio nell'automa in Figura 2.1 la parola ad è accettata, ma il suo prefisso a non lo è. In generale dunque vale $L_m(G) \subseteq \overline{L_m(G)}$; come caso particolare in questa relazione vale l'eguaglianza se e solo se tutti gli stati sono finali. Inoltre se una parola può essere accettata, allora tale parola e tutti i suoi prefissi possono anche a fortiori essere generati: vale dunque $\overline{L_m(G)} \subseteq L(G)$.

2.3.3 Automi Finiti Non deterministici

Definizione 2.3.5. *Un Automa Finito Non deterministico è una quintupla che si denota $G = (X, E, \Delta, x_0, X_m)$ dove:*

- *X è un insieme finito di stati;*

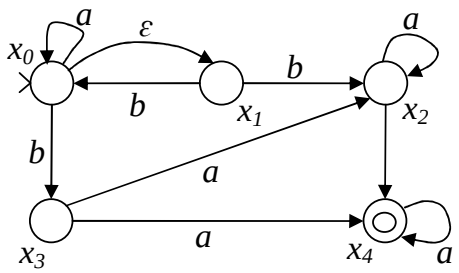

Figura 2.2: Un Automa Finito Non deterministico

- *E è un insieme finito di eventi (cioè un afabeto di simboli);*
- $\Delta : X \times E_\varepsilon \times X$ *è la relazione di transizione, dove* $E_\varepsilon = E \cup \varepsilon$. *Se* $(x, e, \overline{x}) \in \Delta$, *allora a partire dallo stato x ed eseguendo una e-transizione (qui e può essere un simbolo dell'alfabeto oppure la parola vuota) si può raggiungere lo stato* $\overline{x}$*;*
- x_0 *è lo stato iniziale;*
- $X_m \subseteq X$ *è l'insieme di stati finali (o stati marcati).* ■

Anche di un AFN può essere data una rappresentazione del tutto analoga a quella di un AFD.

Esempio 2.3.2. *La Figura 2.2 rappresenta un AFN avente insieme di stati* $X = \{x_0, x_1, x_2, x_3, x_4\}$, *alfabeto* $E = \{a, b\}$, *stato iniziale* x_0 *e insieme di stati finali* $X_m = \{x_4\}$. *La relazione di transizione è data da:*

$$\Delta = \{(x_0, \varepsilon, x_1), (x_0, a, x_0), (x_0, b, x_3), (x_1, b, x_0), (x_1, b, x_2), \\ (x_2, a, x_2), (x_2, b, x_4), (x_3, a, x_2), (x_3, a, x_4), (x_4, a, x_4)\}$$ ■

Gli AFN possono essere visti come una generalizzazione degli AFD. Infatti, la relazione Δ è una generalizzazione della funzione δ e consente di avere:

1. transizioni etichettate con la parola vuota ε (dette anche ε-transizioni). Tali transizioni possono essere viste come corrispondenti ad eventi "silenziosi" che fanno passare da uno stato all'altro senza che essi possano essere osservati;

2. più transizioni uscenti dallo stesso stato e aventi la stessa etichetta. Tali transizioni possono essere viste come corrispondenti a eventi "parzialmente osservabili": cioè si osserva il verificarsi di un evento, ma non si è in grado di determinare esattamente quale, fra due o più eventi etichettati, si è verificato.

Anche per gli AFN è possibile definire una produzione.

Definizione 2.3.6. *Dato un AFN* $G = (X, E, \Delta, x_0, X_m)$ *si definisce produzione una sequenza*

$$x_{j_0} \xrightarrow{e'_1} x_{j_1} \xrightarrow{e'_2} ...x_{j_{k-1}} \xrightarrow{e'_k} x_{j_k}$$

dove per ogni $i = 0, ..., k$ *vale* $x_{j_i} \in X$ *e inoltre da ogni stato* $x_{j_{i-1}}$ *eseguendo la transizione etichettata* e'_i *(si noti che* e'_i *potrebbe essere un evento in E o anche la parola vuota* ε*) si raggiunge lo stato* x_{j_i}*, cioè* $e'_i \in E_\varepsilon$ *e* $(x_{j_{i-1}}, e'_i, x_{j_i}) \in \Delta$ *per ogni i. Si dice anche che tale produzione genera la parola* $w = e'_1 e'_2 ... e'_k$ *partendo dallo stato* x_{j_0} *e raggiungendo lo stato* x_{j_k}*.* ■

Ad esempio una possibile produzione dell'automa in Figura 2.2 è la seguente:

$$x_0 \xrightarrow{a} x_0 \xrightarrow{\varepsilon} x_1 \xrightarrow{b} x_0 \xrightarrow{a} x_0 \xrightarrow{a} x_0$$

Questa produzione raggiunge lo stato x_0 a partire dallo stato x_0 e genera la parola $w = abaa$. Inoltre, in questo caso la lunghezza della parola generata è inferiore a quella della produzione: infatti $|w| = 4$, mentre la produzione contiene cinque transizioni. Si osservi che poichè Δ è una relazione (e non una funzione), possono esistere due produzioni diverse che partono dallo stesso stato e generano la stessa parola. Ad esempio anche

$$x_0 \xrightarrow{a} x_0 \xrightarrow{b} x_3 \xrightarrow{a} x_4 \xrightarrow{a} x_4$$

è una produzione che parte dallo stato x_0 e genera ancora la parola $w = abaa$, raggiungendo però lo stato x_4. E' proprio questa caratteristica, cioè il fatto che a una parola di eventi generata a partire da uno stato dato non corrisponde univocamente una produzione, che fa definire tali automi non deterministici.

La nozione di parola accettata da un AFN va considerata con particolare attenzione, perchè, a causa del non determinismo, alla stessa parola possono essere

associate diverse produzioni.

Definizione 2.3.7. *Dato un AFN $G = (X, E, \Delta, x_0, X_m)$ si dice che una parola $w \in E^*$ è*

1. *generata da G se esiste $x \in X$ tale che $(x_0, w, x) \in \Delta^*$, cioè se esiste una produzione che genera w a partire dallo stato iniziale;*
2. *accettata (o marcata) da G se esiste $x \in X_m$ tale che $(x_0, w, x) \in \Delta^*$, cioè se esiste una produzione che genera w a partire dallo stato iniziale raggiungendo uno stato finale.* ■

Si noti che a causa del non determinismo possono esistere più stati raggiunti generando w a partire dallo stato iniziale. La parola w è accettata se almeno uno fra questi è finale. Ad esempio nel caso dell'automa in Figura 2.2 si è già visto che la parola $w = abaa$ può essere generata (fra le altre) dalle due produzioni

$$x_0 \xrightarrow{a} x_0 \xrightarrow{\varepsilon} x_1 \xrightarrow{b} x_0 \xrightarrow{a} x_0 \xrightarrow{a} x_0$$
$$x_0 \xrightarrow{a} x_0 \xrightarrow{b} x_3 \xrightarrow{a} x_4 \xrightarrow{a} x_4$$

La prima non porta a uno stato finale. Tuttavia poichè la seconda porta allo stato x_4 che è finale, si può concludere che la parola $abaa$ è accettata.

Definizione 2.3.8. *Dato un AFN $G = (X, E, \Delta, x_0, X_m)$ è possibile associare ad esso due linguaggi:*

- *Il linguaggio generato*

  $$L(G) = \{w \in E^* \mid \exists x \in X \in : (x_0, w, x) \in \Delta^*\},$$

  cioè il linguaggio che costituisce l'insieme di tutte le parole generate.

- *Il linguaggio accettato (o marcato)*

  $$L(G) = \{w \in E^* \mid \exists x \in X_m : (x_0, w, x) \in \Delta^*\} \subseteq L(G),$$

  cioè il linguaggio che costituisce l'insieme di tutte le parola accettate. ■

Anche per gli AFN come già visto per il caso degli AFD, valgono le seguenti inclusioni: $L_m(G) \subseteq \overline{L_m(G)} \subseteq L(G) = \overline{L(G)}$.

2.3.4 Reti di Petri

Il modello di *reti di Petri* che viene richiamato è quello delle *reti Posto/Transizione*. Si tratta di un modello che non permette di rappresentare la temporizzazione degli eventi ma solo l'ordine in cui questi si verificano. Le reti P/T sono importanti per diversi motivi:

- sono un formalismo grafico e matematico, conseguentemente, combinano alla facilità di comprensione, la possibilità di applicare un vasto insieme di tecniche di analisi per lo studio delle proprietà di interesse;
- consentono di rappresentare in maniera compatta anche sistemi con uno spazio di stato di dimensione elevata. Esse infati non richiedono di rappresentare tutti i possibili valori dello stato di un sistema ma solo le regole che ne governano l'evoluzione;
- permettono la rappresentazione esplicita del concetto di concorrenza;
- permettono la rappresentazione modulare, ossia consentono di rappresentare ogni sottosistema con una sottorete, e successivamente combinare le diverse sottoreti per ottenere il modello del sistema complessivo.

Una rete P/T è un grafo bipartito e pesato. I due tipi di vertici sono detti: *posti* (rappresentati da cerchi) e *transizioni* (rappresentate da barre o da rettangoli).

Definizione 2.3.9. *Una* rete Posto/Transizione *è una struttura* $N = (P, T, Pre, Post)$ *dove:*

- $P = \{p_1, p_2, ..., p_m\}$ *è l'insieme degli* m posti.
- $T = \{t_1, t_1, ..., t_n\}$ *è l'insieme delle* n transizioni.
- $Pre : P \times T \longrightarrow \mathbb{N}$ *è la* funzione di pre-incidenza *che specifica gli archi diretti dai posti alle transizioni (detti archi pre) e viene rappresentata mediante una matrice* $m \times n$. *Più esattamente,* $Pre(p, t)$ *indica quanti archi vanno dal posto* p *alla transizione* t; *come caso particolare vale* $Pre(p, t) = 0$ *se non esistono archi da* p *a* t.
- $Post : P \times T \longrightarrow \mathbb{N}$ *è la* funzione di post-incidenza *che specifica gli archi diretti dalle transizioni ai posti (detti archi post) e viene rappresentata mediante una matrice* $m \times n$. *Più esattamente,* $Post(p, t)$ *indica quanti archi vanno dalla transizione* t *al posto* p; *come caso particolare vale* $Post(p, t) = 0$ *se non esistono archi da* t *a* p.

Si suppone che $P \cap T = \emptyset$ (cioè posti e transizioni sono insiemi disgiunti) e che $P \cup T \neq \emptyset$ (cioè la rete è costituita da almeno un posto o da una transizione). ■

Le matrici Pre e $Post$ sono delle matrici di interi non negativi. Si denota con $Pre(\cdot, t)$ la colonna della matrice Pre relativa alla transizione t, e con $Pre(p, \cdot)$ la riga della matrice Pre relativa al posto p. La stessa notazione vale per la matrice $Post$. L'informazione sulla struttura di rete contenuta nelle matrici Pre e $Post$ può essere compattata in un'unica matrice, detta di *incidenza*.

Definizione 2.3.10. *Data una rete $N = (P, T, Pre, Post)$, con m posti ed n transizioni, la* matrice di incidenza $C : P \times T \longrightarrow Z$ *è la matrice $m \times n$ definita come:*

$$C = Post - Pre$$

cioè il generico elemento di C vale $C(p, t) = Post(p, t) - Pre(p, t)$. ■

Nel compattare le due matrici Pre e $Post$ per costruire la matrice di incidenza, spesso si perde qualche informazione sulla struttura della rete. Data C non è sempre possibile ricostruire il grafo, mentre date le matrici Pre e $Post$ è possibile ricostruire perfettamente il grafo. Vediamo un esempio che renderà più chiare le considerazioni precedenti.

Esempio 2.3.3. *In Figura 2.3 è rappresentata la rete $N = (P, T, Pre, Post)$ con un insieme dei posti pari a $P = \{p_1, p_2, p_3, p_4\}$ e un insieme delle transizioni $T = \{t_1, t_2, t_3, t_4, t_5\}$. La matrice Pre vale:*

$$Pre = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

mentre la matrice $Post$ vale:

$$Post = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

La matrice di incidenza vale:

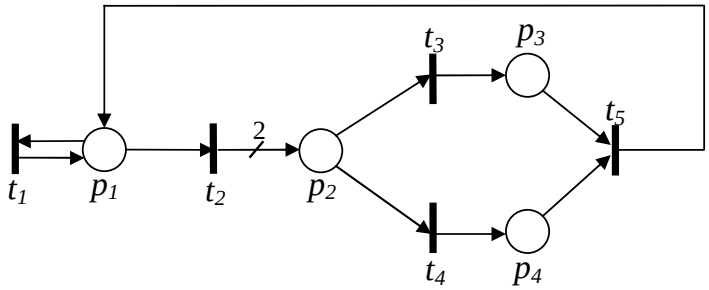

Figura 2.3: Una rete P/T

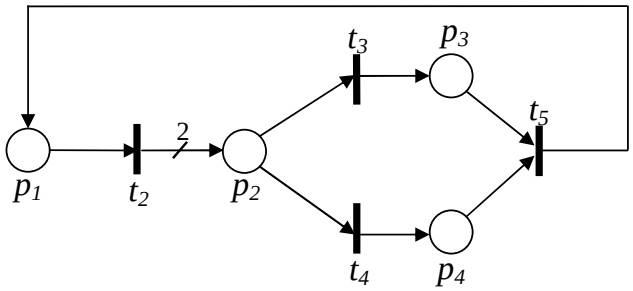

Figura 2.4: Una rete P/T ricostruita a partire dalla sua matrice di incidenza

$$C = \begin{pmatrix} 0 & -1 & 0 & 0 & 1 \\ 0 & 2 & -1 & -1 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & -1 \end{pmatrix}$$

Si noti che $Post(p_2, t_2) = 2$ e dunque vi sono due archi che vanno dalla transizione t_2 al posto p_2. Nella figura, invece di rappresentare i due archi è usata una notazione semplificata che consiste nel rappresentare un solo arco avente per etichetta un numero (2 in questo caso) che indica la sua molteplicità. Si noti in Figura 2.4 come, ricostruendo la RdP a partire dalla matrice di incidenza, si vadano a perdere tutte le informazioni relative ad eventuali cappi. ■

Infine, data una transizione si definiscono i seguenti sistemi di posti:

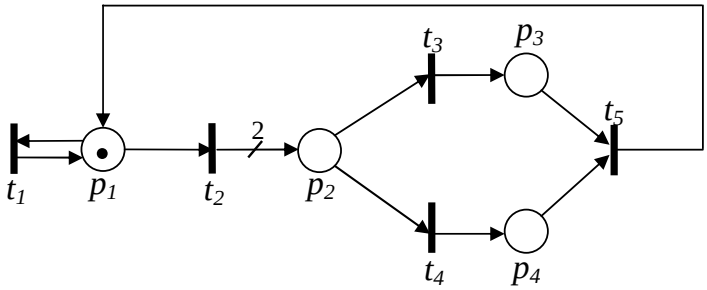

Figura 2.5: Evoluzione di una rete marcata. Marcatura iniziale.

${}^\bullet t = \{p \in P \mid Pre(p,t) > 0\}$: è l'insieme dei posti in ingresso a t.
$t^\bullet = \{p \in P \mid Post(p,t) > 0\}$: è l'insieme dei posti in uscita da t.
${}^\bullet p = \{t \in T \mid Post(p,t) > 0\}$: è l'insieme delle transizioni in ingresso a p.
$p^\bullet = \{t \in T \mid Pre(p,t) > 0\}$: è l'insieme delle transizioni in uscita da p.

Ad esempio, nella rete in Figura 2.3 vale ${}^\bullet t_2 = \{p_1\}$, $t_2^\bullet = \{p_2\}$, ${}^\bullet p_2 = \{t_2\}$, $p_2^\bullet = \{t_3, t_4\}$.

Un concetto fondamentale che caratterizza le *reti di Petri* è la marcatura, tramite essa è possibile definire lo stato di una rete P/T.

Definizione 2.3.11. *Una* marcatura *è una funzione* $M : P \longrightarrow N$ *che assegna ad ogni posto un numero intero non negativo di marche (o gettoni) rappresentate graficamente con dei pallini neri dentro i posti.* ■

Considerando l'esempio in Figura 2.3, una marcatura possibile M è $M(p_1) = 1$, $M(p_2) = M(p_3) = M(p_4) = 0$ come mostrato in Figura 2.5. Un'altra marcatura possibile è quella mostrata in Figura 2.6, dove $M(p_1) = 0, M(p_2) = 2, M(p_3) = M(p_4) = 0$.

Definizione 2.3.12. *Una rete* N *con una marcatura iniziale* M_0 *è detta* rete marcata *o* sistema di rete*, e viene indicata come* $\langle N, M_0 \rangle$. ■

Una rete marcata è in effetti un *sistema ad eventi discreti* a cui è associato un comportamento dinamico.

Vediamo cosa si intende per scatto di una transizione.

Definizione 2.3.13. *Una* transizione t *è* abilitata *dalla marcatura* M *se*

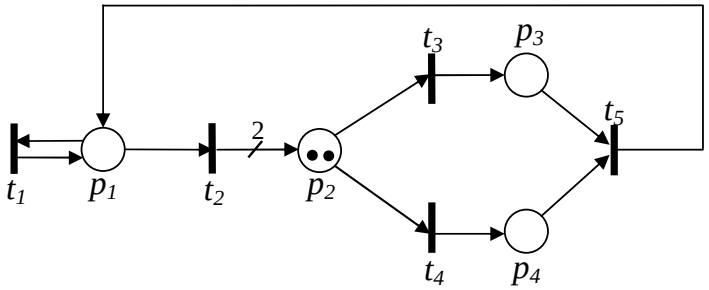

Figura 2.6: Evoluzione di una rete marcata. Marcatura raggiunta dopo lo scatto della transizione t_2.

$$M \geqslant Pre(\cdot, t)$$

cioè se ogni posto $p \in P$ della rete contiene un numero di marche pari o superiore a $Pre(p, t)$. Per indicare che t è abilitata da M si scrive $M[t\rangle$. Per indicare che t' non è abilitata da M si scrive . ■

Definizione 2.3.14. *Una transizione t abilitata da una marcatura M può* scattare. *Lo* scatto *di t rimuove $Pre(p, t)$ marche da ogni posto $p \in P$ e aggiunge $Post(p, t)$ in ogni posto $p \in P$, determinando una nuova marcatura M'. Cioé vale:*

$$M' = M - Pre(\cdot, t) + Post(\cdot, t) = M + C(\cdot, t)$$

Scriviamo $M[t\rangle M'$ per indicare che lo scatto di t da M determina la marcatura M'. ■

Definizione 2.3.15. *Una sequenza di transizioni $\sigma = t_{j_1} t_{j_2} ... t_{j_r} \in T^*$ è abilitata da una marcatura M, se: la transizione t_{j_1} è abilitata da M e il suo scatto porta ad $M_1 = M + C(\cdot, t_{j_1})$; la transizione t_{j_2} è abilitata da M_1 e il suo scatto porta ad $M_2 = M_1 + C(\cdot, t_{j_2})$; ecc. Una sequenza abilitata σ viene anche detta* sequenza di scatto *e ad essa corrisponde la* traiettoria*: $M[t_{j_1}\rangle M_1[t_{j_2}\rangle M_2 ... [t_{j_r}\rangle M_r$.* ■

Per indicare che la sequenza σ è abilitata da M si scrive $M[\sigma\rangle$. Per indicare che lo scatto di σ da M determina la marcatura M' si scrive $M[\sigma\rangle M'$. Ad esempio, nella rete in Figura 2.6 una possibile sequenza di transizioni abilitata dalla marcatura data è $\sigma = t_3 t_4 t_5 t_1$ il cui scatto porta alla marcatura iniziale $M_0 = [\ 1 \quad 0 \quad 0 \quad 0\ ]^T$.

Definizione 2.3.16. *Il* comportamento *(o* linguaggio*) di una rete marcata $\langle N, M_0\rangle$ è l'insieme delle sequenze di scatto abilitate dalla marcatura iniziale, cioè l'insieme:*

$$L(N, M_0) = \{\sigma \in T^* \mid M_0[\sigma\rangle\}.$$

T^* è l'insieme di tutte le possibili sequenze di transizioni facenti parte dell'insieme T. ■

Definizione 2.3.17. *Una marcatura M è detta* raggiungibile *in $\langle N, M_0\rangle$ se esiste una sequenza di scatto σ tale che $M_0[\sigma\rangle M$. L'*insieme di raggiungibilità *di una rete marcata $\langle N, M_0\rangle$ è l'insieme delle marcature che possono venir raggiunte a partire dalla marcatura iniziale, cioè l'insieme:*

$$R(N, M_0) = \{M \in N^m \mid \exists\sigma \in L(N, M_0) : M_0[\sigma\rangle M\}.$$

■

Infine,

Definizione 2.3.18. *Sia $\langle N, M_0\rangle$ una rete marcata e C sia la sua matrice di incidenza. Se M è raggiungibile da M_0 scattando la sequenza di transizioni σ vale:*

$$M = M_0 + C \cdot \vec{\sigma}.$$

$\vec{\sigma}$ è detto vettore di scatto e ha tante componenti quante sono le transizioni. ■

Una *rete di Petri* che non ha cicli orientati è detta *aciclica*.

La *funzione di etichettatura* $\mathscr{L} : T \longrightarrow L \cup \{\varepsilon\}$ assegna a ogni transizione $t \in T$ o un simbolo appartenente a un dato alfabeto L o la stringa vuota ε:

$$T_u = \{t \in T | L(t) = \varepsilon\}.$$

Le transizioni appartenenti all'insieme T_u sono chiamate non osservabili o silenziose. Le transizioni appartenenti all'insieme T_o sono dette osservabili.

2.4 Diagnosi mediante automi

In questa sezione viene trattata la diagnosi di un *Sistema ad Eventi Discreti* mediante l'approccio con gli automi, il quale è stato studiato ed esposto da Lafortune, docente presso l'università del Michigan, in collaborazione con diversi suoi studenti di dottorato, tra cui Sampath *et al.* [15]. La diagnosi di un guasto nei (SED) si sviluppa in due principali fasi: la costruzione del modello del SED che deve essere diagnosticato, seguito dalla costruzione del "protocollo diagnostico", in altre parole l'insieme di regole impiegate per la scoperta e la localizzazione del guasto.

Il comportamento del sistema è modellato con un linguaggio regolare ed è rappresentato da un automa a stati finiti. In breve, un linguaggio è detto diagnosticabile se è possibile scoprire, con un ritardo finito, il verificarsi di certi eventi non osservabili, detti eventi difettosi o guasti. Una procedura sistematica per la scoperta e l'isolamento dei guasti consiste nell'uso dei diagnosticatori. Il *diagnosticatore* è una macchina a stati finiti costruita a partire dal modello di macchina a stati finiti del sistema. Questa macchina esegue la diagnosi osservando il comportamento *on-line* del sistema. Gli stati del diagnosticatore portano informazioni sul guasto e il verificarsi di questo può essere scoperto (con un ritardo finito) esaminando questi stati. Le condizioni necessarie e sufficienti affinchè un linguaggio sia diagnosticabile sono basate sul diagnosticatore. Quindi, il diagnosticatore ha due scopi:

- la verifica *off-line* delle proprietà di diagnosticabilità del sistema;
- la scoperta e la localizzazione *on-line* dei guasti.

Il sistema da diagnosticare è modellato come un automa a stati finiti:

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

dove X è lo spazio di stato, E è l'insieme degli eventi, $\delta : X \times E \longrightarrow X$ è la funzione di transizione parziale e x_0 è lo stato iniziale del sistema. Il modello G rende conto sia del comportamento normale, che di quello di guasto del sistema.

Un automa non deterministico è caratterizzato da eventi non osservabili. Le ε-transizioni di un automa non deterministico sono eventi che si verificano nell'automa, ma che non sono visibili, o osservabili dall'esterno. Questa perdita di osservabilità è dovuta all'assenza di un sensore che rileva l'occorrenza dell'e-

vento o al fatto che l'evento accade in una locazione remota, ma non è comunicato al sito centrale; questa è una tipica situazione nei sistemi di informazione distribuiti. Gli eventi di guasto che non causano un immediato cambiamento nella lettura del sensore sono un esempio di eventi non osservabili. Invece che etichettare tutte le transizioni dovute ad eventi non osservabili con ε e ottenere un automa non deterministico come modello del sistema, definiamo gli eventi di queste transizioni come "autentici", ma caratterizziamo questi eventi come non osservabili.

In altre parole, il nostro modello del sistema sarà un automa deterministico il cui insieme degli eventi è diviso in due insiemi disgiunti:

$$E = E_o \cup E_u,$$

dove E_o rappresenta l'insieme degli eventi osservabili e E_u quello degli eventi non osservabili.

Consideriamo $E_f \subseteq E$ l'insieme degli eventi di guasto che devono essere diagnosticati. Assumiamo, senza perdità di generalità, che $E_f \subseteq E_u$, dal momento che un guasto osservabile può essere facilmente diagnosticato. L'obiettivo è quello di identificare il manifestarsi dei guasti, se ne accadono, date le tracce di eventi osservati dal sistema, in cui ci sono solo gli eventi osservabili appartenenti a E_o. Dividiamo l'insieme dei guasti in insiemi disgiunti non vuoti corrispondenti ai diverti tipi di guasto:

$$E_f = E_{f_1} \cup E_{f_2} \cup ... \cup E_{f_m}.$$

Indichiamo con $\prod_f = \{1, 2, ..., m\}$ questa partizione. Essa è motivata dalle seguenti considerazioni:

- una strumentazione inadeguata può rendere impossibile la diagnosi del singolo guasto;
- non possiamo pretendere di identificare singolarmente il verificarsi di ogni singolo evento di guasto. Possiamo solamente essere interessati a conoscere se ne è accaduto uno, appartenente a un insieme in cui l'effetto dell'insieme dei guasti sul sistema è lo stesso. Quando si dice che *"un guasto di tipo F_i si è verificato"* significa che si è verificato qualche evento appartenente all'insieme E_{f_i}.

Faremo le seguenti assunzioni sul sistema da analizzare:

1. il linguaggio L generato da G è vivo, ossia esiste una transizione per ogni stato $x \in X$; in altre parole il sistema può raggiungere uno stato morto nel quale nessun evento è possibile;
2. non esiste in G alcun ciclo di eventi non osservabili.

La prima assunzione, che riguarda la vivezza del linguaggio, è fatta per motivi di semplicità; mentre la seconda assicura che le osservazioni avvengano con una certa regolarità. Dal momento che la scoperta dei guasti è basata sulle transizioni osservabili del sistema, richiediamo che il sistema non generi sequenze arbitrariamente lunghe di eventi non osservabili.

Introduciamo la nozione di *proiezione*.

Definizione 2.4.1. *Definiamo la* proiezione $P : E^* \longrightarrow E_o^*$ *dove:*

(i) $P(\varepsilon) = \varepsilon$;

(ii) $P(\sigma) = \sigma, \quad se\ \sigma \in E_o$;

(iii) $P(\sigma) = \varepsilon, \quad se\ \sigma \in E_u$;

(iv) $P(s\sigma) = P(s)P(\sigma), \quad s \in E^*,\ \sigma \in E.$ ■

P "cancella" semplicemente gli eventi non osservabili in una traccia.

Esempio 2.4.1. *Consideriamo la stringa $s = ab\varepsilon_1 a\varepsilon_2$, dove l'insieme degli eventi osservabili $E_o = \{a, b\}$ e l'insieme degli eventi non osservabili $E_u = \{\varepsilon_1, \varepsilon_2\}$. La proiezione è pari a $P(s) = aba$.* ■

2.4.1 Il diagnosticatore

Per il problema di diagnosi dei guasti abbiamo già detto che considereremo, per le ragioni sopra spiegate, i guasti come eventi non osservabili. Abbiamo quindi che il modello del sistema è rappresentato da un automa non deterministico, dove compaiono delle transizioni non osservabili. Un automa non deterministico G_{nd} può essere sempre convertito in un automa deterministico che ha un linguaggio equivalente, cioè un automa che genera e accetta lo stesso linguaggio dell'automa G_{nd}. Lo spazio di stato dell'automa deterministico equivalente sarà

un sottinsieme dell' insieme potenza dello spazio di stato dell'automa non deterministico. Ciò implica che se l'automa non deterministico è a stati finiti, allora l'equivalente automa deterministico sarà pure a stati finiti.

Introduciamo ora il *diagnosticatore* che è un nuovo automa a stati finiti costruito a partire dal modello del sistema $G = (X, E, \delta, x_0)$. Questa macchina è usata per diagnosticare *on-line* il comportamento di G. Il diagnosticatore è anche usato per verificare le condizioni necessarie e sufficienti per la diagnosticabilità. Quindi in molte applicazioni dove il modello del sistema contiene eventi non osservabili potremmo essere interessati a determinare se alcuni eventi non osservabili *potrebbero essere accaduti* o *devono essere accaduti* nella stringa di eventi eseguita dal sistema. Questo è il problema della *diagnosi di eventi*. Se questi eventi non osservabili di interesse modellano dei guasti dei componenti del sistema, allora la conoscenza che uno di questi eventi sia accaduto è molto importante quando monitoriamo le prestazioni del sistema. Il punto chiave è che più noi continuiamo a osservare il comportamento del sistema, più possiamo ridurre l'incertezza sui prefissi delle stringhe di eventi eseguiti dal sistema. La procedura per la costruzione del diagnosticatore può essere riassunta nel seguente modo: l'idea di base è di partire dallo stato x_0 di G e considerare tutti gli stati che vengono raggiunti da questo con transizioni osservabili e non osservabili, e identifichiamo poi tutti gli eventi in E che sono abilitati da uno o più stati in x_0 e mettiamo tali eventi in un unico insieme attivo di x_0. Per ogni evento contenuto nell'insieme attivo, identifichiamo tutti gli stati in X che possono essere raggiunti a partire da uno stato in x_0, per semplicità supponiamo per il momento che l'insieme E_f contenga un solo evento di guasto, allora:

(a) attacchiamo l'etichetta N agli stati che possono essere raggiunti da x_0 con stringhe non osservabili, quindi appartenenti all'insieme E_u ma non appartenenti all'insieme E_f;

(b) attacchiamo l'etichetta Y agli stati che possono essere raggiunti da x_0 con stringhe non osservabili che contengono almeno una occorrenza dell'evento di guasto;

(c) se uno stato z può essere raggiunto sia eseguendo un evento di guasto che non eseguendolo, allora nello stato iniziale di $Diag(G)$ crea: zN e zY.

Una volta costruiti gli altri stati raggiungibili del diagnosticatore occorre attaccare le etichette agli stati di G nel modo sopra indicato. Inoltre, qualunque stato raggiungibile dallo stato zY deve mantenere l'etichetta Y ad indicare che l'evento di guasto si è verificato raggiungendo z e quindi anche nel processo di

raggiungimento del nuovo stato. Per il diagnosticatore non è definito alcun insieme di stati marcati. Ricapitolando, $Diag(G)$ ha come insieme di eventi E_o, è un automa deterministico e genera il linguaggio $L(Diag(G)) = P[L(G)]$. Ogni stato di $Diag(G)$ è un sottoinsieme di $X \times \{N, Y\}$. Facciamo un esempio per chiarire meglio.

Esempio 2.4.2. *Consideriamo l'automa G in Figura 2.7. L'insieme di eventi non osservabili è pari a* $E_u = \{e_d, u, v\}$*, mentre l'insieme degli eventi osservabili è* $E_o = \{a, b, c, d, g\}$*. Dopo che osserviamo la stringa* $t = a$ *non sappiamo se il sistema ha eseguito l'evento non osservabile* e_d *o meno. Però dopo che osserviamo la stringa* $s = tg = ag$*, sappiamo con certezza che l'evento non osservabile deve essere accaduto. In questo modo possiamo diagnosticare il verificarsi dell'evento non osservabile* e_d *dopo che abbiamo osservato s. Il diagnosticatore è un automa a stati finiti che tiene traccia del verificarsi degli eventi non osservabili, diagnosticando, se possibile, la loro occorrenza. Indichiamo il diagnosticatore con* $Diag(G)$ *o* G_{diag}*. Per semplicità assumiamo di voler diagnosticare un solo evento non osservabile* $e_d \in E_u$*. Se volessimo diagnosticare più di un evento, possiamo o costruire un diagnosticatore per ogni evento che deve essere diagnosticato o costruire un singolo diagnosticatore che simultaneamente tiene traccia di tutti gli eventi non osservabili. Verranno usate due tipi di etichette:* N *per indicare* "No, e_d non si è ancora verificato" *e* Y *per indicare* "Si, e_d si è verificato". *Quando attacchiamo un'etichetta allo stato dell'automa* $x \in X$*, scriveremo* xN *o* xY *per abbreviare rispettivamente la notazione* (x, N) *o* (x, Y)*.* ■

La diagnosi *on-line* dell'evento di guasto è condotta tenendo traccia del corrente stato del diagnosticatore in risposta agli eventi osservati eseguiti dal sistema G. Possiamo avere tre tipi di stato nel diagnosticatore:

- se tutti gli stati di G nello stato corrente di $Diag(G)$ hanno etichetta N, allora siamo sicuri che l'evento di guasto non si è ancora verificato. Chiameremo uno stato di questo tipo uno *stato negativo*;
- se tutti gli stati di G nello stato corrente di $Diag(G)$ hanno etichetta Y, allora siamo sicuri che l'evento di guasto si è verificato ad un certo punto nel passato. Chiameremo uno stato di questo tipo uno *stato positivo*;
- se lo stato corrente di $Diag(G)$ contiene almeno uno stato di G con etichetta N e almeno uno stato di G con etichetta Y, allora l'evento di guasto potrebbe o meno essersi verificato precedentemente. Chiameremo uno stato di questo tipo uno *stato incerto*.

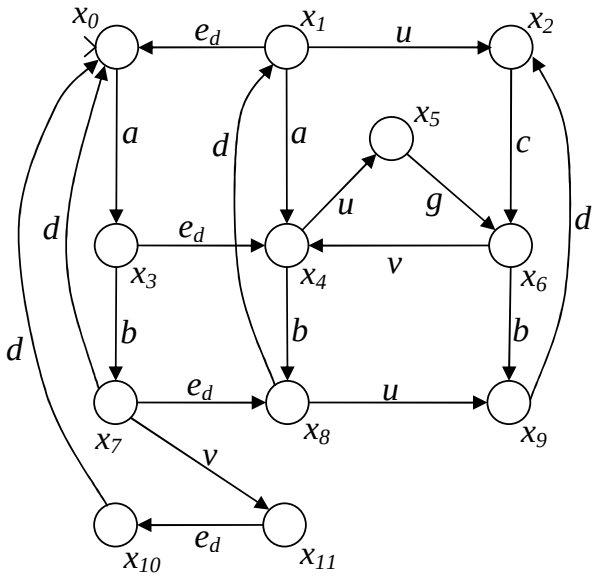

Figura 2.7: Automa G introdotto nell'esempio 2.4.2

Vediamo un esempio.

Esempio 2.4.3. *La Figura 2.8 mostra il diagnosticatore G_{diag} dell'automa G mostrato in Figura 2.7 dove l'evento e_d è l'evento che deve essere diagnosticato. Il diagramma di transizione dello stato G_{diag} mostra che dopo che viene osservato l'evento c o l'evento g siamo sicuri che l'evento e_d si è verificato dal momento che tutti gli stati di G che compaiono negli stati del diagnosticatore hanno la stessa etichetta Y. Mentre potremmo non sapere esattamente lo stato di G, sappiamo con certezza che l'evento e_d è accaduto. D'altra parte, se la stringa osservata non contiene nessun evento c o g, siamo negli stati di G_{diag} dove abbiamo stati sia con l'etichetta N che con l'etichetta Y. Quindi, anche se è possibile che l'evento e_d sia accaduto, non siamo certi sulla sua occorrenza. Possiamo trarre conclusioni sul verificarsi dell'evento e_d dall'esame degli stati del diagnosticatore.* ■

Formalizziamo la nozione di diagnosticabilità. E' semplice dare la definizione di diagnosticabilità per linguaggi che sono vivi, cioè linguaggi che non contengono stringhe di terminazione: un linguaggio L è detto vivo se per qualunque $s \in L$ esiste un e tale che $se \in L$. In termini di automa, un automa che non ha stati di deadlock genera un linguaggio vivo. L'evento non osservabile appartenente

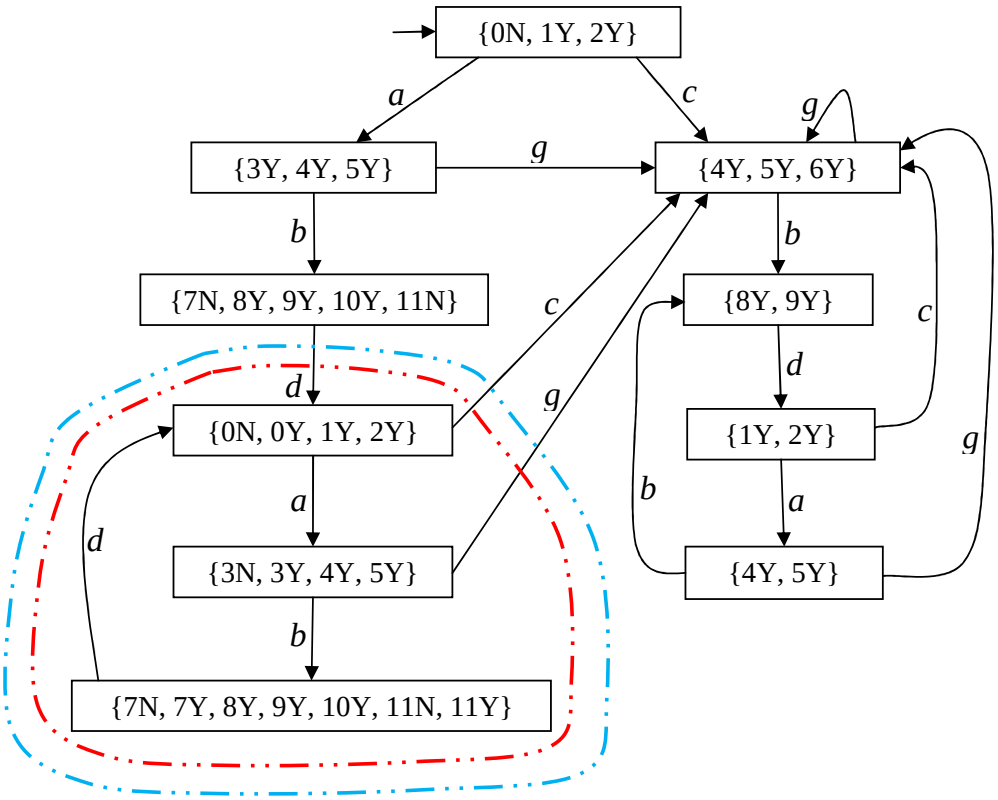

Figura 2.8: Diagnosticatore G_{diag} dell'esempio 2.4.3 dell'automa G in Figura 2.7 introdotto nell'esempio 2.4.2

all'insieme E_f è non diagnosticabile nel linguaggio vivo $L(G)$ se esistono due stringhe s_N e s_Y in $L(G)$ che soddisfano le seguenti condizioni:

(i) s_Y contiene l'evento di guasto ed s_N no;

(ii) s_Y è arbitrariamente lunga dopo l'evento di guasto;

(iii) $P(s_N) = P(s_Y)$.

Quando non esiste una coppia di stringhe come quella appena definita, l'evento di guasto è detto *diagnosticabile* in $L(G)$.

In generale, possiamo associare ad un ciclo di stati incerti in $Diag(G)$ due cicli in G, uno che include solo stati con etichetta Y negli stati incerti e uno contenente

solo stati con etichetta N negli stati incerti, chiameremo tale ciclo in $Diag(G)$ un *ciclo indeterminato*. Per definizione la presenza di un ciclo indeterminato implica una violazione della diagnosticabilità. Viceversa, una violazione della diagnosticabilità genererà la presenza di un ciclo indeterminato in $Diag(G)$, dal momento che $Diag(G)$ è deterministico ed ha uno spazio di stato finito. Questo ci permette di concludere che la proprietà di diagnosticabilità può essere testata analizzando i cicli di stati incerti in $Diag(G)$. Se uno qualunque di questi è indeterminato, allora la diagnosticabilità è violata.

Esempio 2.4.4. *Riconsideriamo l'automa G in Figura 2.7 e il suo diagnosticatore in Figura 2.8 e consideriamo le seguenti due stringhe: $s_N = (abd)^m$ e $s_Y = e_d(abd)^m$, questo vuol dire che la stringa "s_Y è arbitrariamente lunga dopo e_d". In altre parole, il suffisso di s_Y dopo e_d cicla in un loop di G. Chiaramente $P(s_Y) = P(s_N)$. Così, se s_Y è eseguito dal sistema, allora non saremo mai in grado di diagnosticare con certezza l'occorrenza di e_d: dopo il verificarsi di e_d, $P(s_Y) = P(s_N)$ ciclerà negli stati incerti del diagnosticatore G_{diag}. Quando ciò accade, diciamo che il manifestarsi di e_d in s_Y è* non diagnosticabile. *Quando non esiste una coppia di stringhe come quella sopra definita, e_d è detto* diagnosticabile *in* L(G). *Nel caso in cui la proprietà di diagnosticabilità sia soddisfatta, siamo sicuri che l'evento e_d si è verificato, quindi $Diag(G)$ entrerà in uno stato positivo in un numero finito di eventi dopo il verificardi di e_d. Per vedere ciò, osserviamo che:*

(i) G non ha cicli di eventi non osservabili dopo e_d (per ipotesi);

(ii) $Diag(G)$ non può ciclare in un ciclo di stati incerti in quanto questo contradirrebbe la diagnosticabilità;

(iii) $Diag(G)$ ha un insieme di stati finito.

Quando il diagnostiscatore *ha un ciclo di stati incerti, potenzialmente potrei non essere certo sul verificarsi del guasto e_d per un numero di eventi (osservabili) arbitrariamente lungo se questo non esce mai dal ciclo. Se riusciamo a trovare due stringhe s_Y ed s_N dove $e_d \in s_Y$, $e_d \notin s_N$, $P(s_N) = P(s_Y)$, e $P(s_Y)$ entra e non esce mai dal ciclo di stati incerti del diagnosticatore, rileviamo una violazione della diagnosticabilità. Osservando le Figure 2.7 e 2.8 si può notare che possiamo associare al ciclo di stati incerti in G_{diag} (nella Figura 2.8 il ciclo è cerchiato in rosso) il ciclo "$x_1 \longrightarrow x_4 \longrightarrow x_8 \longrightarrow x_1$" in G (vedi Figura 2.7), ed inoltre qualunque stringa che rimane in questo ciclo in G deve contenere l'evento e_d dal momento che questi stati hanno etichetta "Y" in G_{diag}. Possiamo anche associare a questo ciclo di stati incerti in G_{diag} (nella Figura 2.8 il ciclo è*

cerchiato in celeste) il ciclo "$x_0 \longrightarrow x_3 \longrightarrow x_7 \longrightarrow x_0$" in G (vedi Figura 2.7), ed inoltre qualunque stringa che rimane in questo ciclo in G non deve contenere l'evento e_d dal momento che questi stati hanno etichetta "N" in G_{diag}. Possiamo usare questi due cicli in G per costruire due stringhe s_Y ed s_N che soddisfano le condizioni precedentemente enunciate e quindi violano la diagnosticabilità. ■

E'importante sottolineare che la presenza di un ciclo di stati incerti in un diagnosticatore non implica necessariamente il fatto di non riuscire a non diagnosticare il verificarsi di un evento di guasto. Consideriamo il seguente esempio.

Esempio 2.4.5. *Consideriamo il sistema e il diagnosticatore in Figura 2.9 dove e_d, l'evento che deve essere diagnosticato, è l'unico evento non osservabile del sistema. Questo diagnosticatore ha un ciclo di stati incerti (vedi nella Figura 2.9 il ciclo cerchiato in rosso). L'unico caso tale per cui il diagnosticatore rimanga nel ciclo di stati incerti, inidicato in rosso nella figura, è il ciclo "$x_6 \longrightarrow x_{10} \longrightarrow x_{11} \longrightarrow x_6$", ma questi stati hanno tutti etichetta N nei corrispondenti stati del diagnosticatore. Perciò il ciclo di stati incerti del diagnosticatore non è indeterminato. Grazie all'assenza di cicli indeterminati possiamo dire che nel sistema il verificarsi dell'evento e_d è sempre diagnosticabile. Infatti, se l'evento e_d si manifesta, il diagnosticatore lascerà il ciclo di stati incerti ed entrerà nello stato $5Y$ con l'osservazione dell'evento t. In questo esempio, il fatto che il sistema possa ciclare negli stati x_6, x_{11} e x_{10}, causando il fatto che il diagnosticatore cicli nel suo ciclo di stati incerti, non è visto come una perdita di diagnosticabilità, dal momento che le stringhe che causano questo "ciclare" non contengono e_d. C'è da notare come questo approccio dia la possibilità non solo di dire se il sistema sia diagnosticabile o meno, ma se è diagnosticabile possiamo dire con quale ritardo scopriamo che il guasto si è verificato, ossia dopo quanti passi dall'evento di guasto siamo in grado di dire che il guasto si è verificato. Il numero di passi, ossia il numero di transizioni che dobbiamo osservare dopo il guasto per scoprire la sua occorrenza, viene calcolato facilmente a partire dal sistema. In questo esempio, il numero di passi dopo il guasto che mi fanno capire che il guasto si è verificato è* 6 *(il suffisso della stringa dopo e_d è "$bgdbgt$").* ■

In altre parole, la diagnosticabilità, richiede che ogni evento di guasto conduca a delle osservazioni distinte, sufficienti per permettere unicamente l'identificazione del guasto con un ritardo finito. Il caso di guasti multipli dello stesso tipo, ossia appartenenti allo stesso insieme della partizione, richede particolare attenzione. Quando più di un guasto dello stesso tipo, detto f_i, si manifesta lungo una traccia s di L, la definizione sopra data di diagnosticabilità, non richiede che ognuna di queste occorrenze sia scoperta. E' sufficiente essere capaci di conclu-

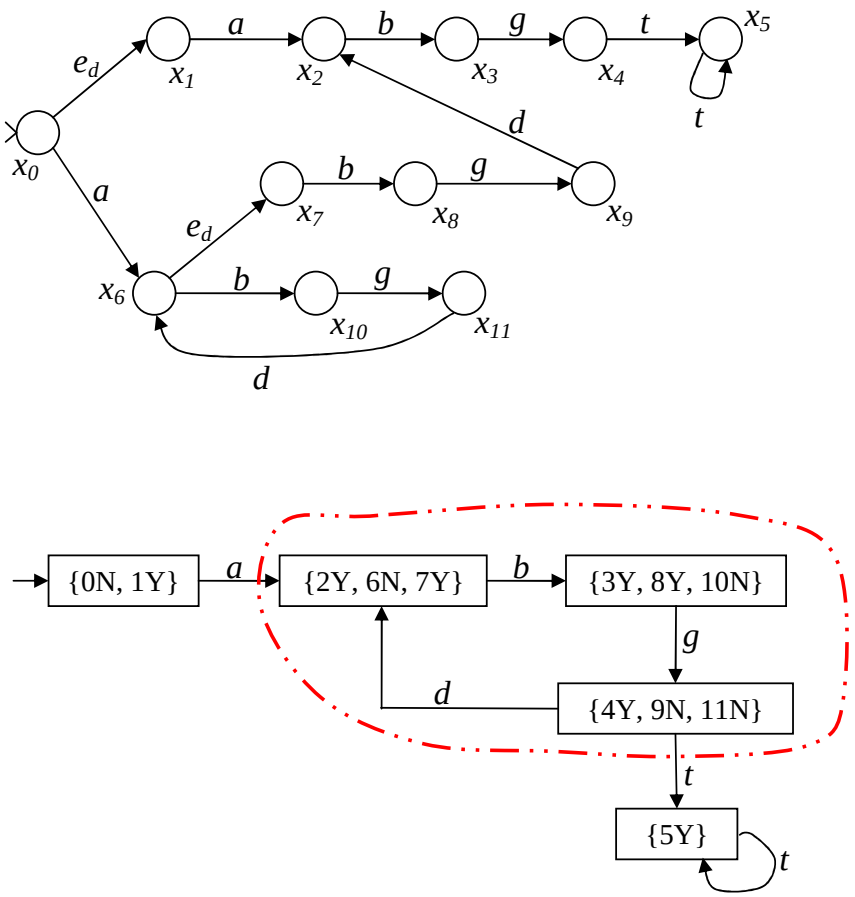

Figura 2.9: Sistema e corrispondente diagnosticatore per l'evento non osservabile e_d considerati nell'esempio 2.4.5

dere, dopo un numero limitato di eventi dopo il verificarsi del primo guasto, che lungo s, un guasto dell'insieme E_{f_i} è avvenuto.

Facciamo un esempio.

Esempio 2.4.6. *Consideriamo il sistema rappresentato in Figura 2.10. L'insieme degli eventi osservabili è* $E_o = \{a, b, c, d\}$*, l'insieme degli eventi non osservabili è* $E_u = \{g_1, g_2, g_3, z\}$*, e l'insieme contenenti i guasti è* $E_f = \{g_1, g_2, g_3\}$*, infatti* $E_f \subseteq E_u$*. Consideriamo che lo stato iniziale del sistema sia* x_0*. Se scegliamo le partizioni:*

$$E_{f_1} = \{g_1, g_2\},\ E_{f_2} = \{g_3\}$$

non è richiesto che si debba distinguere tra i guasti g_1 *e* g_2*. Il diagnosticatore per questo primo sistema è mostrato in Figura 2.11. Il sistema illustrato è diagnosticabile con un numero di eventi* $n_1 = 2$ *e* $n_2 = 1$*. Infatti considerate le parole* $w_1 = bg_1g_2c$*,* $w_2 = bg_1zc$*,* $w_3 = bg_1g_3d$*, in tutte possiamo identificare come traccia che termina in un evento di guasto, la traccia* $s = bg_1$*, quindi poichè* n_1 *è il numero massimo di transizioni del sistema in cui si può scoprire il verificarsi di un guasto dopo s, risulta* $n_1 = 2$*. Allo stesso modo per l'insieme* E_{f_2} *possiamo considerare la parola* $w_1 = bg_1g_3d$*. La traccia s in questo caso è pari a* $s = bg_1g_3$*, conseguentemente il numero massimo di transizioni del sistema in cui si può scoprire il verificarsi di un guasto dopo s risulta* $n_2 = 1$*. Il fatto che il sistema sia diagnosticabile è facilmente osservabile anche dal diagnosticatore (vedi Figura 2.11), in quanto non contiene cicli di stati incerti, gli unici tre cicli sono infatti relativi ad uno stato negativo (cappio con l'evento* a*, nella figura in rosso) e due stati positivi (cappi con gli eventi* c *e* d*, nella figura in celeste). D'altra parte se la partizione fosse stata:*

$$E_{f_1} = \{g_1\},\ E_{f_2} = \{g_2\},\ E_{f_3} = \{g_3\},$$

allora il sistema sarebbe stato non diagnosticabile dal momento che non è possibile determinare il manifestarsi dell'evento g_2*. Il diagnosticatore per questo caso è mostrato in Figura 2.12. Se, infatti, osservo la stringa* bc *non sono in grado di dire se il guasto* g_2 *si è verificato o meno, perchè i due eventi potrebbero appartenere indifferentemente alle due parole* $w_1 = bg_1g_2c$ *e* $w_2 = bg_1zc$*. Quindi* g_2 *non è diagnosticabile perchè non conduce a delle osservazioni distinte, sufficienti per permettere unicamente l'identificazione del guasto con un ritardo finito. Questo fatto è sottolineato anche dal ciclo indeterminato (nella Figura 2.12 in rosso). Tale ciclo, rappresentato dal cappio nello stato* $\{3Y1, 3Y1Y2\}$*, è indeterminato in quanto il guasto appartenente alla classe* E_{f_2} *compare una volta*

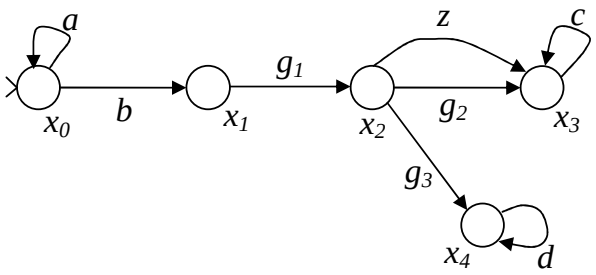

Figura 2.10: Sistema con guasti multipli

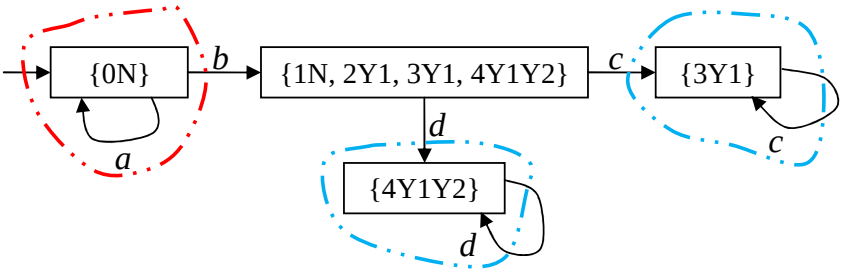

Figura 2.11: Diagnosticatore con due classi di guasto: $E_{f_1} = \{g_1, g_2\}$, $E_{f_2} = \{g_3\}$ riferito al sistema in Figura 2.10.

con l'etichetta N (infatti $3Y1$ *equivale a scrivere:* $3Y1N2N3$*) e una volta con l'etichetta Y (in* $3Y1Y2$*).* ■

2.5 Diagnosi mediante reti di Petri

In tale sezione si presenta un efficiente approccio per la diagnosi dei guasti che utilizza le reti di Petri. Tale approccio è stato proposto da Cabasino *et al.* in [14].

Nel seguito, così come proposto in [13], affrontiamo la diagnosi dei guasti dei *sistemi ad eventi discreti* modellati dalle reti posto/transizione. In particolare, viene riportato brevemente l'approccio alla diagnosi dei SED mediante le reti di Petri etichettate, presentato in [13] da Cabasino *et al.*. La procedura proposta è basata su precedenti risultati ottenuti sulle RdP non etichettate e permette di considerare, in aggiunta, quegli eventi che sono indistinguibili, ossia gli even-

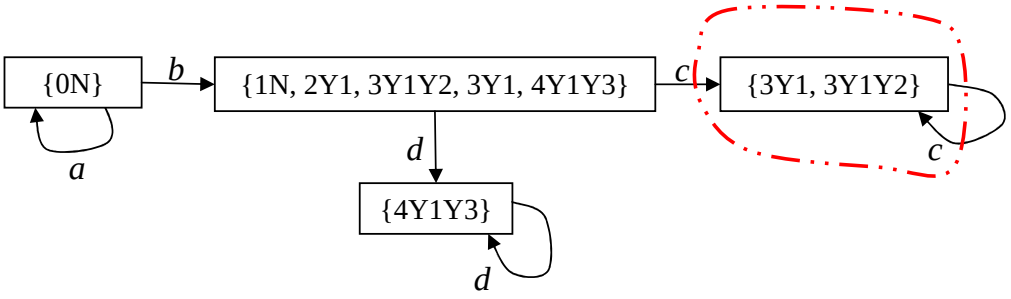

Figura 2.12: Diagnosticatore con tre classi di guasto: $E_{f_1} = \{g_1\}$, $E_{f_2} = \{g_2\}$, $E_{f_3} = \{g_3\}$ riferito al sistema in Figura 2.10.

ti che producono un segnale di uscita che è osservabile, ma è comune ad altri. Verranno analizzate tecniche che portano alla diagnosi dei guasti studiate nelle tesi di laurea di Pocci [28] e di Perria [26]. In particolare, assumiamo che i guasti siano modellati da transizioni non osservabili, ma che possano anche esistere altre transizioni che rappresentano un comportamento legale, ma che sono ugualmente non osservabili. Quindi assumiamo che l'insieme delle transizioni possa essere diviso in $T = T_o \cup T_u$, dove T_o è l'insieme delle transizioni osservabili, e T_u è l'insieme delle transizioni non osservabili. Quando una transizione osservabile scatta osserviamo la sua etichetta, quindi le nostre osservazioni consistono in sequenze di simboli nell'alfabeto L. L'insieme di transizioni non osservabili è suddiviso a sua volta in due sottoinsiemi, detti $T_u = T_f \cup T_{reg}$, dove T_f include tutte le transizioni di guasto mentre T_{reg} include tutte le transizioni relative a eventi non osservabili ma regolari. L'insieme T_f è ulteriormente suddiviso in r sottoinsiemi T_f^*, dove $i = 1, ..., r$, che modellano le differenti classi di guasto.

Definizione 2.5.1. *Un sistema di rete $\langle N, M_0 \rangle$ è detto diagnosticabile rispetto alla classe di guasto T_f^i se non esistono due sequenze σ_1 e σ_2 in T^* che soddisfano le seguenti condizioni:*

- $\mathscr{L}(\sigma_1) = \mathscr{L}(\sigma_2)$, *ovvero le sequenze hanno la stessa proiezione osservabile;*
- $\forall\, t_f \in T_f^i$, $t_f \notin \sigma_1$, *ovvero σ_1 non contiene transizioni di guasto nella classe di guasto T_f^i;*
- $\exists$ *almeno una* $t_f \in T_f^i$ *tale che* $t_f \in \sigma_2$,

- *σ_2 è arbitrariamente lunga dopo l'occorrenza del guasto $t_f \in T_f^i$, cioè esiste almeno una decomposizione $\sigma_2 = \sigma_2' t_f \sigma_2''$ tale che dato un qualsiasi $k \in \mathbb{N}$ possiamo scegliere σ_2'' in modo che $|\sigma_2''| > k$.* ■

Definizione 2.5.2. *Un sistema di rete $\langle N, M_0 \rangle$ è detto diagnosticabile se è diagnosticabile in riferimento a tutte le classi di guasto.* ■

Si noti che la diagnosticabilità di un sistema non implica che siamo in grado di distinguere tra transizioni appartenenti alla medesima classe di guasto, ma semplicemente, implica che se una o più transizioni in una data classe di guasto sono scattate, allora dopo un numero finito di osservazioni siamo in grado di stabilire che almeno una transizione di quella classe è scattata. Ci occuperemo di fornire le condizioni necessarie e sufficienti per la diagnosticabilità, in particolare consideriamo sistemi di RdP etichettate che soddisfano le seguenti condizioni:

1. Il sistema di rete $\langle N, M_0 \rangle$ è *limitato*, e *non si blocca dopo lo scatto di una transizione di guasto*;
2. La sottorete non osservabile è aciclica;
3. La funzione di etichettatura $\mathscr{L} : T \to L \cup \{\varepsilon\}$ può associare la stessa etichetta a differenti transizioni;
4. La struttura di N è nota così come la sua marcatura iniziale M_0.

Richiamiamo alcuni definizioni di base introdotte per la prima volta nei lavori [14] e [13].

Definizione 2.5.3. *[14] Data una marcatura M ed una transizione $t \in T_o$ definiamo*

$$\Sigma_{\min}(M,t) = \{\sigma \in T_u^* \mid M[\sigma\rangle M',\ M' \geq Pre(\cdot,t),\ \nexists\, \sigma' \mid M[\sigma'\rangle M'',\ M'' \geq Pre(\cdot,t) :\ \pi(\sigma') \lneqq \pi(\sigma)\}$$

l'insieme delle spiegazioni minime *di t ad M e definiamo*

$$Y_{\min}(M,t) = \pi(\Sigma_{\min}(M,t))$$

il corrispondente insieme degli e-vettori minimi. ■

Nel caso delle RdP etichettate quello che osserviamo è l'etichetta l. Allora è utile definire i seguenti insiemi.

Definizione 2.5.4. *[13] Data una marcatura M ed una osservazione* $\mathrm{l} \in L$ *definiamo l'* insieme delle spiegazioni minime di l a M *come*

$$\hat{\Sigma}_{\min}(M,t) = \cup_{t \in T_l} \cup_{\sigma \in \Sigma_{\min}(M,t)} \{(t,\sigma)\},$$

l'insieme delle coppie (transizione etichettata l - corrispondente e-vettore minimo) e definiamo l'insieme degli e-vettori minimi di l a M come

$$\hat{Y}_{\min}(M,l) = \cup_{t \in T_l} \cup_{e \in \Sigma_{\min}(M,t)} \{(t,\sigma)\},$$

l'insieme delle coppie (transizione etichettata l - corrispondente e-vettore minimo). ■

Data una parola $w \in L^*$ chiamiamo *giustificazione di w* la corrispondente sequenza di transizioni non osservabili intervallate con σ_0 il cui scatto abilita σ_0 e il cui vettore di scatto è minimo.

Definizione 2.5.5. *[13] Dato il sistema di rete $\langle N, M_0 \rangle$ con funzione di etichettatura $\mathscr{L} = T \to L \cup \{\varepsilon\}$, dove $N = (P, T, Pre, Post)$ e $T = T_o \cup T_u$. Sia $w \in L^*$ una data osservazione. Definiamo*

$$\hat{\mathscr{J}}(w) = \{\, (\sigma_o, \sigma_u),\ \sigma_o \in T_o^*,\ \mathscr{L}(\sigma_o) = w,\ \sigma_u \in T_u^* \mid [\exists \sigma \in \mathscr{S}(w) : \sigma_o = P_o(\sigma),\ \sigma_u = P_u(\sigma)] \wedge [\nexists \sigma' \in \mathscr{S}(w) : \sigma_o = P_o(\sigma'),\ \sigma'_u = P_u(\sigma') \wedge \pi(\sigma'_u) \lneq \pi(\sigma_u)] \,\}$$

l'insieme di coppie (sequenza $\sigma_o \in T_o^*$ con $\mathscr{L}(\sigma_o) = w$ - corrispondente giustificazione *di w). Definiamo*

$$\hat{Y}_{\min}(M_0, w) = \{(\sigma_o, y), \sigma_o \in T_o^*,\ \mathscr{L}(\sigma_o) = w,\ y \in \mathbb{N}_u^n \mid \exists (\sigma_o, \sigma_u) \in \hat{\mathscr{J}}(w) : \pi(\sigma_u) = \vec{y}\}$$

l'insieme di coppie (sequenza $\sigma_o \in T_o^*$ con $\mathscr{L}(\sigma_o) = w$ – corrispondente j-vettore*).* ■

In altre parole, $\hat{\mathscr{J}}(w)$ è l'insieme di coppie sequenza $\sigma_o \in T_o^*$ etichettata w - giustificazione e i vettori di scatto di queste sequenze sono detti *j-vettori*.

Definizione 2.5.6. *[13] Dato il sistema di rete $\langle N, M_0 \rangle$ con funzione di etichettatura $\mathscr{L} : T \to L \cup \{\varepsilon\}$, dove $N = (P, T, Pre, Post)$ e $T = T_o \cup T_u$. Sia w una data osservazione e $(\sigma_o, \sigma_u) \in \hat{\mathscr{J}}(w)$ una generica coppia (sequenza di osservazioni etichettate w - corrispondente giustificazione minima). La marcatura*

$$M_b = M_0 + C_u \cdot \vec{y} + C_o \cdot \vec{y'}, \qquad \vec{y} = \pi(\sigma_u),\ \vec{y'} = \pi(\sigma_o),$$

raggiunta scattando σ_o intervallata con la giustificazione minima σ_u, è detta marcatura di base *e $\vec{y}$ è chiamato* j-vettore *(o vettore giustificazione).* ■

In altre parole, la marcatura di base M_b è la marcatura raggiunta dalla marcatura M_0 con lo scatto di w e di tutte quelle transizioni non osservabili che sono strettamente necessarie ad abilitare w.

Ovviamente, poiché in generale esiste più di una giustificazione per una parola w, la marcatura di base non è in genere unica.

Definizione 2.5.7. *[13] Dato il sistema di rete $\langle N, M_0\rangle$ con funzione di etichettatura $\mathscr{L} = T \to L \cup \{\varepsilon\}$, dove $N = (P, T, Pre, Post)$ e $T = T_o \cup T_u$. Sia $w \in L^*$ una parola osservata. Definiamo*

$$\mathscr{M}(w) = \big\{(M, y) \mid (\exists \sigma \in \mathscr{S}(w) \ : \ M_0[\sigma\rangle M) \ \wedge (\exists (\sigma_o, \sigma_u) \in \hat{\mathscr{J}}(w) \ : \\ \sigma_o = P_o(\sigma), \sigma_u = P_u(\sigma), \ y = \pi(\sigma_u))\big\}$$

l'insieme di coppie (marcatura di base - relativo j-vettore) che sono consistenti con $w \in L^*$. ■

Esempio 2.5.1. *Consideriamo la rete in Figura 2.13 dove l'insieme di transizioni osservabili è $T_o = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$ e l'insieme di transizioni non osservabili è $T_u = \{\varepsilon_8, \varepsilon_9, \varepsilon_{10}, \varepsilon_{11}, \varepsilon_{12}, \varepsilon_{13}, \varepsilon_{14}, \varepsilon_{15}\}$. La funzione di etichettatura è $\mathscr{L}(t_1) = a$, $\mathscr{L}(t_2) = \mathscr{L}(t_3) = b$, $\mathscr{L}(t_4) = \mathscr{L}(t_5) = c$ e $\mathscr{L}(t_6) = \mathscr{L}(t_7) = d$.*

Assumiamo $w = abb$. L'insieme di giustificazioni è $\hat{\mathscr{J}}(w) = \{(t_1 t_2 t_3, \varepsilon_8\varepsilon_9\varepsilon_{12}), (t_1 t_2 t_2, \varepsilon_8\varepsilon_9\varepsilon_{10}\varepsilon_{11})\}$ e l'insieme di j-vettori è $\hat{Y}_{min}(M_0, w) = \{(t_1 t_2 t_3, [1\ 1\ 0\ 0\ 1\ 0\ 0\ 0]^T), (t_1 t_2 t_2, [1\ 1\ 1\ 1\ 0\ 0\ 0\ 0]^T)\}$. I j-vettori precedenti conducono alle marcature $M_1 = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]^T$ e $M_2 = [0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T$. Allora $\mathscr{M}(w) = \{(M_1, [1\ 1\ 0\ 0\ 1\ 0\ 0\ 0]^T)), (M_2, [1\ 1\ 1\ 1\ 0\ 0\ 0\ 0]^T))\}$. ■

Introduciamo ora la definizione di diagnosticatore:

Definizione 2.5.8. *[13] Un diagnosticatore è una funzione*

$$\Delta : L^* \times \{T_f^1, T_f^2, \ldots, T_f^r\} \to \{0, 1, 2, 3\}$$

che associa a ciascuna osservazione $w \in L^*$ ed a ciascuna classe di guasto T_f^i, $i = 1, \ldots, r$, uno stato di diagnosi.

- $\Delta(w, T_f^i) = 0$ *se per tutte le $\sigma \in \mathscr{S}(w)$ e per tutte le $t_f \in T_f^i$ si verifica che $t_f \notin \sigma$.*

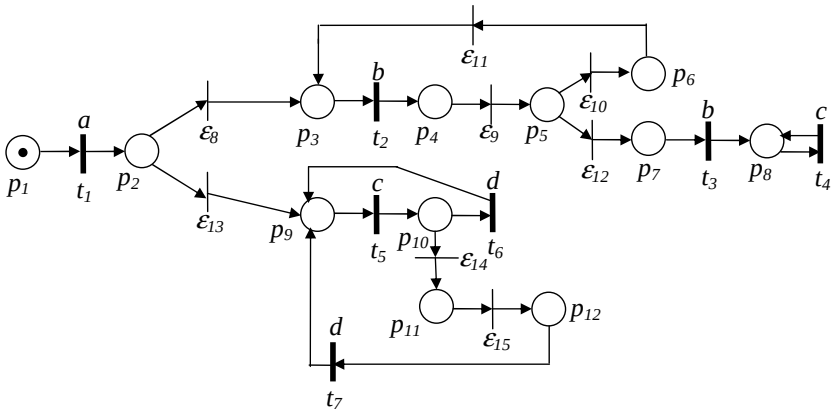

Figura 2.13: Esempio di rete di Petri

In questo caso, non si è verificato alcun guasto associato alla i-esima classe di guasto, in quanto nessuna delle sequenze di scatto consistenti con l'osservazione contiene una transizione di guasto della i-esima classe.

- $\Delta(w, T_f^i) = 1$ *se:*

  (i) esiste una $\sigma \in \mathscr{S}(w)$ *e una* $t_f \in T_f^i$ *tali che* $t_f \in \sigma$ *ma*

  (ii) per tutte le $(\sigma_o, \sigma_u) \in \hat{\mathscr{J}}(w)$ *e per tutte le* $t_f \in T_f^i$ *si ha che* $t_f \notin \sigma_u$.

  In questo caso una transizione di guasto della classe i-esima può essersi verificata ma non è contenuta in nessuna giustificazione di w.

- $\Delta(w, T_f^i) = 2$ *se esistono due coppie* $(\sigma_o, \sigma_u), (\sigma'_o, \sigma'_u) \in \hat{\mathscr{J}}(w)$ *tali che:*

  (i) esiste $t_f \in T_f^i$ *tale che* $t_f \in \sigma_u$*;*

  (ii) per tutte le $t_f \in T_f^i$, $t_f \notin \sigma'_u$.

  In questo caso una transizione di guasto della classe i è contenuta in una giustificazione di w, ma non in tutte.

- $\Delta(w, T_f^i) = 3$ *se per tutte le* $\sigma \in \mathscr{S}(w)$ *esiste una* $t_f \in T_f^i$ *tale che* $t_f \in \sigma$.

  In questo caso, un guasto associato alla i-esima classe di guasto deve essersi verificato, poiché tutte le sequenze scattabili consistenti con l'osservazione contengono almeno un guasto in T_f^i. ■

La precedente definizione può essere espressa come proposizione in termini di marcatura di base e di giustificazione.

Proposizione 2.5.1. *[13] Consideriamo la parola osservata* $w \in L^*$.

- $\Delta(w, T_f^i) \in \{0,1\}$ *se e solo se per tutte le coppie* $(M, y) \in \mathscr{M}(w)$ *e per tutte* $t_f \in T_f^i$ *si verifica che* $y(t_f) = 0$.
- $\Delta(w, T_f^i) = 2$ *se e solo se esiste una* $(M, y) \in \mathscr{M}(w)$ *e* $(M', y') \in \mathscr{M}(w)$ *tale che:*
  (i) esiste $t_f \in T_f^i$ *tale che* $y(t_f) > 0$,
  (ii) per tutte $t_f \in T_f^i$, $y'(t_f) = 0$.
- $\Delta(w, T_f^i) = 3$ *se e solo se per tutte le coppie* $(M, y) \in \mathscr{M}(w)$ *esiste* $t_f \in T_f^i$ *tale che* $y(t_f) > 0$.

La seguente proposizione introduce un metodo utilizzato per distinguere tra gli stati di diagnosi 0 e 1, basato sulla risoluzione di un sistema di equazioni lineari.

Proposizione 2.5.2. *[13] Per una RdP la cui sottorete non osservabile sia aciclica, sia* $w \in L^*$ *una parola osservata tale che per tutte le* $(M, y) \in \mathscr{M}(w)$ *si abbia* $y(t_f) = 0 \ \forall \ t_f \in T_f^i$.

Consideriamo l'insieme di vincoli:

$$\mathscr{T}(M) \quad = \quad \begin{cases} M + C_u \cdot z \geq \vec{0}, \\ \sum\limits_{t_f \in T_f^i} z(t_f) > 0, \\ z \in \mathbb{N}^{n_u}. \end{cases} \tag{2.1}$$

- $\Delta(w, T_f^i) = 0$ *se* $\forall \ (M, y) \in \mathscr{M}(w)$ *l'insieme di vincoli* (2.1) *non ammette soluzione.*
- $\Delta(w, T_f^i) = 1$ *se* $\exists \ (M, y) \in \mathscr{M}(w)$ *tale che l'insieme di vincoli* (2.1) *ammette soluzione.*

In base ai precedenti risultati, se la sottorete non osservabile è aciclica, la diagnosi può essere effettuata semplicemente osservando l'insieme $\mathscr{M}(w)$ per qualunque parola w osservata e, nel caso in cui lo stato di diagnosi vale 0 o 1,

valutando in aggiunta se il corrispondente insieme di vincoli interi 2.1 ammette una soluzione.

Esempio 2.5.2. *Consideriamo la RdP in Figura 2.13, dove* $T_f^1 = \{\varepsilon_{12}\}$ *e* $T_f^2 = \{\varepsilon_{14}\}$.

Sia $w = abb$. $\mathscr{M}(w) = \{(M_1, [1\ 1\ 0\ 0\ 1\ 0\ 0\ 0]^T), (M_2, [1\ 1\ 1\ 1\ 0\ 0\ 0\ 0]^T)\}$, *dove* $M_1 = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]^T$ *e* $M_2 = [0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T$ *sono le marcature determinate nell'esempio 2.5.1. In questo caso si ha ad esempio che,* $\Delta(w, T_f^1) = 2$. *Infatti,* $y_1 = [1\ 1\ 0\ 0\ 1\ 0\ 0\ 0]^T$ *contiene* ε_{12} *appartenente a* T_f^1, *mentre* $y_2 = [1\ 1\ 1\ 1\ 0\ 0\ 0\ 0]^T$ *non contiene una transizione di guasto contiene* $t_f \in T_f^1$. ■

In [13] è stato mostrato che nel caso di RdP limitate e etichettate un utile strumento per la diagnosi è il *Basis Reachability Graph* (BRG).

In [13] è stato mostrato come il BRG necessita di essere modificato se lo si vuole utilizzare come strumento ausiliario per stabilire se il sistema è diagnosticabile. A questo scopo è stato definito un nuovo grafo, chiamato *Modified Basis Reachability Graph* (MBRG). Nella tesi di Pocci [28] la diagnosticabilità veniva valutata utilizzando un diagnosticatore chiamato *Basis Reachability Diagnoser* (BRD). Si tratta di un grafo deterministico che usato in aggiunta al MBRG permette di ricavare le condizioni necessarie e sufficienti per la diagnosticabilità. Nella tesi di Perria [26] si è passati alla costruzione del *Basis Reachability Diagnoser Modified* (MBRD), che consente di determinare in unione al MBRG le condizioni necessarie e sufficienti per la diagnosticabilità, con un miglioramento dal punto di vista delle prestazioni rispetto all'approccio visto nella tesi di Pocci [28]. Poichè la verifica della diagnosticabilità è basata sui concetti di cicli incerti e indeterminati. Per la ricerca dei cicli incerti all'interno del MBRD si è fatto uso dell'algoritmo di Johnson [23]. Nel seguito di questo Capitolo verranno illustrati brevemente il BRG, il MBRG, il BRD, il MBRD. Tutti questi ultimi sono dei grafi, la cui costruzione è affidata ai toolbox realizzati da Pocci [28] e Perria [26].

2.6 Primo approccio di diagnosticabilità

2.6.1 Grafo di Raggiungibilità di Base

Il BRG (*Basis Reachability Graph*) è un grafo deterministico che ha tanti nodi quante sono il numero di marcature di base raggiungibili. A ciascun nodo è associata una distinta marcatura di base M ed un vettore riga con tanti elementi

quante sono il numero di classi di guasto. Tali elementi possono assumere solo valori binari: 1 se $\mathscr{T}(M)$ l'insieme di vincoli 2.1 ammette soluzioni, 0 altrimenti.

Gli archi sono etichettati con eventi osservabili di L ed e-vettori. In particolare, esiste un arco da un nodo contenente una marcatura di base M ad un nodo contenente la marcatura di base M' se e solo se esiste una transizione t per la quale esiste una spiegazione a partire da M e lo scatto di t e di una delle sue spiegazioni minime porta ad M'. L'arco che porta da M ad M' è etichettato $(\mathscr{L}(t), e)$, dove $e \in Y_{min}(M, t)$ ed $M' = M + C_u \cdot e + C(\cdot, t)$.

Il numero di nodi del BRG è sempre finito, poichè l'insieme delle marcature di base è un sottoinsieme dell'insieme di raggiungibilità, che è finito essendo la rete limitata. In aggiunta, il vettore riga di valori binari associati al nodo del BRG ci permette di fare distinzione tra lo stato di diagnosi 1 e 0. L'algoritmo per la determinazione del BRG è riportato in [28], in breve, l'algoritmo costruisce il BRG a partire dal nodo iniziale a cui corrisponde la marcatura iniziale ed un vettore binario, che definisce quali classi di guasto possono verificarsi in M_0. Ora, consideriamo tutte le etichette $l \in L$ tali che esista una transizione t con $\mathscr{L}(t) = l$ per cui esiste una spiegazione minima in M_0. Per ciascuna di queste transizioni calcoliamo la marcatura risultante dallo scatto t da $M_0 + C_u \cdot e$, per ciascuna $e \in Y_{min}(M_0, t)$. Se è ottenuta una coppia (marcatura, vettore binario) non contenuta nei nodi precedenti, viene aggiunto un nuovo nodo nel grafo. L'arco che va dal nodo iniziale al nuovo nodo è etichettato (l, e). La procedura è iterata sino a che tutte le marcature di base non sono considerate. L'approccio descritto richiede sempre di enumerare uno spazio di stato che è un sottoinsieme stretto dell'insieme di raggiungibilità.

2.6.2 Grafo di Raggiungibilità di Base Modificato

Poichè il BRG non è sufficiente per effettuare concretamente l'analisi di diagnosticabilità della rete, è necessario modificarlo, se si vuole utilizzare come strumento ausiliario per stabilire se il sistema in analisi sia diagnosticabile, e quindi è stato definito un nuovo grafo, MBRG (*Modified Basis Reachability Graph*). Assumiamo, che l'insieme delle transizioni possa essere diviso in $T = T_o \cup T_u$. L'insieme di transizioni non osservabili è suddiviso a sua volta in due sottoinsiemi, detti $T_u = T_f \cup T_{reg}$, dove T_f include tutte le transizioni di guasto mentre T_{reg} include tutte le transizioni relative a eventi non osservabili ma regolari. L'insieme T_f è ulteriormente suddiviso in r sottoinsiemi T_f^*, dove $i = 1, ..., r$, che modellano le differenti classi di guasto.

Il MBRG è un grafo deterministico i cui nodi contengono due elementi (M, x) :

$M \in \mathbb{N}$ è la marcatura definita come segue ed x è un vettore riga di $\{0,1\}^{|T_f|}$ dove $x(i) = 1$ se $\mathscr{T}(M)$ in 2.1 è ammissibile in riferimento alla i-esima classe, $x(i) = 0$ altrimenti.

La marcatura M del nodo è calcolata, analogamente alla costruzione del BRG, come marcatura di base, assumendo che tutte le transizioni di guasto siano osservabili. Ciò siginifica che le spiegazioni minime sono limitate alle sole transizioni in T_{reg}.

Gli archi possono essere etichettati in due differenti modi, indipendentemente dall'evento in analisi.

Nel caso di eventi corrispondenti allo scatto di transizioni in T_o, l'etichetta contiene tre informazioni riassunte con $(l(t), e)$, dove $l \in L$ è l'etichetta osservata, t è la transizione etichettata l il cui scatto al nodo di ingresso è abilitato dalla sequenza di transizioni regolari con vettore di scatto $e \in T_{min}^{mod}(M, t)$ (insieme degli e-vettori minimi limitati a T_{reg}) e che porta alla marcatura del nodo di uscita.

Nel caso di eventi corrispondenti a transizioni di guasto, l'etichetta contiene solo due informazioni riassunte con (t_f, e), dove $t_f \in T_f$ è la transizione di guasto il cui scatto al nodo di ingresso è abilitato da una sequenza con vettore di scatto $e \in Y_{min}^{mod}(M, t)$ e che porta alla marcatura del nodo di arrivo. L'algoritmo per la costruzione del MBRG è riportato in [28]. In breve, l'algoritmo costruisce il MBRG a partire da un nodo iniziale a cui corrisponde la marcatura iniziale ed al contempo un vettore binario che definisce quale classe di guasto possa scattare in M_0.

2.6.3 Diagnosticatore di Raggiungibilità di Base

Definizione 2.6.1. *Il BRD (*Basis Reachability Diagnoser*) è un grafo deterministico, dove ciascun nodo contiene le informazioni seguenti:*

i) una o più triple (M, x, h)*, dove:*

- M *è una marcatura di base;*
- $x \in \{0,1\}^{|T_f|}$ *è un vettore riga il cui* i*-esimo elemento è pari a* 1*, se* $\mathscr{T}(M)$ *ammette soluzione in riferimento alla* i*-esima classe di guasto, o è pari a* 0 *altrimenti;*
- $h \in \{N, F\}^{|T_f|}$ *è un vettore riga il cui* i*-esimo elemento è pari a N, se raggiungendo* M *da* M_0 *non si è verificato nessun guasto di* T_{f_i}*, o è pari ad F altrimenti;*

ii) r *etichette* Δ_i*, per* $i = 1,...,r$*, che rappresentano lo stato di diagnosi del nodo in riferimento alle* r classi di guasto.

Infine, gli archi sono etichettati con un simbolo di $\mathscr{L}$. ■

Quindi il BRD è un grafo deterministico che, se utilizzato in aggiunta al MBRG, permette di formulare le condizioni necessarie e sufficienti per la diagnosticabilità. Il BRD può essere facilmente determinato a partire dal MBRG. In particolare, i valori di M e x sono leggibili dal MBRG, osservando semplicemente i nodi contenenti le marcature di base.

Il valore di h può essere dedotto, osservando il percorso (o i differenti percorsi nel caso ce ne fosse più di uno) da M_0 al corrispondente valore di M (indicato con $M_0 \rightsquigarrow M$). Se esiste un percorso $M_0 \rightsquigarrow M$ contenente transizioni di guasto della classe i-esima, allora alla coppia (M, x) è associato un valore $h(i) = F$. Se, viceversa, esiste un percorso $M_0 \rightsquigarrow M$ non contenente alcuna transizione di guasto della classe i-esima, allora alla coppia (M, x) è associato un valore $h(i) = N$. Poichè in generale può esistere più di un percorso da M_0 ad M, uno contenente un guasto di T_{f_i} ed uno no, la coppia (M, x) può essere presente due volte nello stesso nodo, ma con $h(i) = F$ ed $h(i) = N$.

Durante la costruzione del BRD, vengono considerate solo le marcature di base e non tutte le marcature del MBRG. Perciò, per ciascuna etichetta $l \in L$ dobbiamo considerare solo le marcature raggiunte tramite lo scatto di $Y_{min}(M, t)$ per tutte le transizioni t tali che $l = L(t)$.

Lo stato di diagnosi per ciascuna classe di guasto è ottenuto per definizione, semplicemente osservando i due ultimi elementi di tutte le triple del nodo. L'algoritmo che spiega i passi per la costruzione del BRD è in [28]. In breve, l'algortimo costruisce il BRD a partire dal nodo iniziale, a cui corrisponde la tripla (M_0, x_0, h_0), dove M_0 ed x_0 corrispondono al nodo iniziale dell'MBRG, ed il terzo elemento è inizialmente posto pari a $h_0 = N^r$. Il suo stato di diagnosi Δ_i è posto a zero se non esiste una sequenza di transizioni osservabili che abilitano una transizione di guasto in T_{f_i} dalla marcatura iniziale, ossia se l'elemento di x_0 associato all'unica (per assunzione) transizione di guasto $t_{f_i} \in T_f^i$ è nullo, altrimenti Δ_i è posto ad uno.

A partire dal nodo iniziale ed osservando il MBRG, si presta attenzione all'insieme delle marcature di base raggiungibili in seguito allo scatto di transizioni con etichetta l in M_0, immediatamente o dopo lo scatto di una o più transizioni di guasto.

Il nuovo nodo sarà composto da tutte le triple (M', x', h') tali che la coppia (M', x') sia raggiunta nel MBRG con lo scatto di transizioni etichettate l in M_0 o con lo scatto di una spiegazione minima, contenente una o più transizioni di guasto, e, successivamente, l'etichetta l considerata; h' è definito considerando h_0 e tutti i percorsi $M_0 \rightsquigarrow M'$ nel MBRG.

Infine, lo stato di diagnosi Δ_i dipende per ciascun nodo dall'elemento i-esimo dei due vettori x ed h di tutte le marcature appartenenti al nodo.

La procedura viene iterata sino a che non vengono esplorati tutti i nodi.

2.6.4 Condizioni necessarie e sufficienti per la diagnosticabilità

Le condizioni necessarie e sufficienti per la diagnosticabilità possono essere verificate tramite l'utilizzo del BRD congiuntamente al MBRG. Queste condizioni sono basate sul concetto di ciclo indeterminato.

Definizione 2.6.2. *Sia γ un ciclo incerto nel BRD con proiezione osservabile $\rho \in L^*$ e sia $p \in L^*$ un percorso dal nodo iniziale ad uno appartenente al suddetto ciclo. Se nell'MBRG esistono due cicli γ_1 e γ_2 soddisfacenti le tre seguenti condizioni:*

(i) la loro proiezione osservabile è pari a ρ;

(ii) esistono due percorsi p_1 e p_2 con proiezione osservabile p, che dal nodo iniziale dell'MBRG abilitino γ_1 e γ_2;

(iii) entrambi γ_1 e p_2 non contengono un guasto in T_f^i. ■

Teorema 2.6.1. *[13] Un sistema di rete $\langle N, M_0 \rangle$ è diagnosticabile in riferimento alla classe T_F^i se e solo se il suo BRD non ha cicli indeterminati in riferimento a T_f^i.* ■

Corollario 2.6.1. *[13] Un sistema di rete $\langle N, M_0 \rangle$ è diagnosticabile se e solo se il suo BRD non ha cicli indeterminati rispetto ad ogni classe di guasto.* ■

Poichè in un grafo un ciclo è sempre associato ad una ed una sola componente fortemente connessa, l'idea iniziale per la realizzazione di un algoritmo di ricerca dei cicli è stata quella di esplorare tutti i possibili percorsi appartenenti a singole componenti. Per questo scopo, si è analizzato l'algoritmo Tarjan [33] di Tarjan. Tale lavoro si basa sulla *Depth First Search* (DFS) a partire dal nodo iniziale del grafo,. In particolare, una volta determinate tutte le componenti

fortemente connesse del grafo la DFS viene applicata a ciascuna di esse. L'algoritmo però benchè possa essere applicato per la determinazine dei cicli nel BRD, non fornisce tutte le informazioni necessarie per la verifica della diagnosticabilità. Infatti, non si ha alcuna informazione relativa a tutti quei percorsi che dal nodo iniziale del grafo portano ad uno dei nodi appartenenti al ciclo. Perciò, l'informazione, legata alla verifica dell'occorrenza del guasto lungo il cammino che porta al ciclo, deve essere comunque reperita per poter ultimare l'analisi. L'approccio risulta più macchinoso per la determinazione dei cicli, però più efficiente per la verifica della diagnosticabilità. Sempre in [28] è riportato un algoritmo che riassume i passi principali per la verifica della diagnosticabilità, che è basato sulle nozioni teoriche di Cabasino *et al.* [13]. L'algoritmo enumera esaustivamente a partire dal nodo iniziale del BRD tutti i percorsi orientati presenti nel grafo e, per ciascuno di essi, tiene memoria della sequenza dei nodi attraversati tramite l'insieme N e della proiezione osservabile associata tramite l'insieme W. Per ciascun percorso esplorato, i nodi attraversati sono memorizzati in N e le osservazioni associate a tali percorsi sono memorizzate in W. Congiuntamente alla crescita della lunghezza dei percorsi, è verificata la presenza di eventuali cicli. Se il ciclo è incerto, viene verificato che sia anche indeterminato. Se la verifica va a buon fine l'algoritmo termina e dichiara non diagnosticabile la classe di guasto in esame. Altrimenti il ciclo è eliminato, così come nel caso in cui il ciclo non fosse incerto.

Si noti che, non appena si è trovato un ciclo indeterminato per una classe di guasto, si può concludere che il sistema non è diagnosticabile per tale classe di guasto. Al contrario, per stabilire se un sistema è diagnosticabile rispetto ad una classe di guasto è necessario esaminare tutti i cicli incerti per tale classe di guasto e mostrare che nessuno è indeterminato.

2.7 Secondo approccio di diagnosticabilità

2.7.1 Diagnosticatore di Raggiungibilità di Base Modificato

Definizione 2.7.1. *Il MBRD (*Modified Basis Reachability Graph*) è un grafo deterministico in cui ciascun nodo contiene le seguenti informazioni:*

- *una o più coppie* (M, h)*, dove:*
  - *M è una marcatura;*

- *$h \in \{N, F\}^{|T_f|}$ è un vettore riga il cui i-esimo elemento è pari a N, se raggiungendo M da M_0 non si è verificato nessun guasto di T_f^i, o è pari ad F altrimenti;*

• *r etichette Δ_i, per $i = 1, \ldots, r$, che rappresentano lo stato di diagnosi del nodo in riferimento alle r classi di guasto.*

Gli archi sono etichettati con un simbolo in L. ■

Il nodo iniziale del MBRD comprende la marcatura iniziale del MBRG più tutte quelle marcature raggiunte a partire da questa con lo scatto di transizioni di guasto.

A partire da ciascuna marcatura in questo nodo verifichiamo quali etichette sono abilitate, presa una data etichetta il nodo successivo comprenderà tutte le marcature raggiunte con lo scatto dell'etichetta a partire da ogni marcatura del nodo precedente, più tutte quelle marcature che possono essere raggiunte con lo scatto di transizioni di guasto.

A differenza del BRD ogni nodo non è costituito dalle sole marcature di base ma dalle possibili marcature presenti nel MBRG. In [26] è riportato l'algoritmo per il calcolo del MBRD.

Proposizione 2.7.1. *Nel MBRD non esistono nodi con stato di diagnosi pari a* 1 *($\Delta = 1$).* ■

Dimostrazione. Gli stati con $\Delta = 1$ sono quelli che contengono le marcature di base raggiunte senza lo scatto del guasto, ma da cui lo scatto del guasto è possibile. Per costruzione i nodi del MBRD contengono anche tutte le marcature raggiunte con lo scatto di transizioni di guasto, di conseguenza il caso con stato di diagnosi pari a 1 non è presente.

2.7.2 Condizioni necessarie e sufficienti per la diagnosticabilità

Le condizioni necessarie e sufficienti per la diagnosticabilità basate sui concetti di cicli incerti e indeterminati possono essere verificate usando il MBRD in unione con il MBRG. La prima cosa da fare è verificare se il MBRD contiene un ciclo incerto, che risulta essere un potenziale ciclo indeterminato, e successivamente utilizzare il MBRG per verificare se tale ciclo è indeterminato oppure no.

Definizione 2.7.2. *Sia γ un ciclo nell'MBRD con proiezione osservabile $\rho \in L^*$. Il ciclo γ è incerto in riferimento alla classe di guasto T_f^i se e solo se include stati con $\Delta_i = 2$.* ■

Definizione 2.7.3. *Sia γ un ciclo incerto nel MBRD con proiezione osservabile $\rho \in L^*$. Il ciclo γ è indeterminato in riferimento alla classe di guasto T_f^i se nel MBRG esistono due cicli γ_1 e γ_2 che soddisfano le seguenti condizioni:*

(i) *la loro proiezione osservabile è pari a ρ;*

(ii) *γ_2 contiene un guasto in T_f^i, mentre γ_1 non contiene un guasto in T_f^i. .* ■

In altre parole affinché un ciclo incerto nel MBRD sia indeterminato, deve esistere nel MBRG almeno un ciclo N, ossia un ciclo composto da nodi del MBRG con valore del vettore $\vec{h}$, nel nodo del MBRD, pari a N e almeno un ciclo F, ossia un ciclo costituito da nodi del MBRG con valore del vettore $\vec{h}$ nel nodo del MBRD, pari a F, relativamente alla classe di guasto in esame. Una volta individuato un ciclo incerto nel MBRD per la classe di guasto i-esima, per la determinazione dei cicli N e dei cicli F si procede in questo modo:

- CICLI N: per ogni nodo del ciclo incerto trovato nel MBRD si individuano al suo interno tutte le marcature (corrispondenti a nodi del MBRG) con valore del vettore $\vec{h}$ pari a N, per la classe di guasto in esame. A questo punto i cicli N vengono ottenuti come combinazioni dei nodi con $h = N$ individuati in ciascun nodo del ciclo incerto;
- CICLI F: per ogni nodo del ciclo incerto trovato nel MBRD si individuano al suo interno tutte le marcature (corrispondenti a nodi del MBRG) con valore del vettore $\vec{h}$ pari a F, per la classe di guasto in esame. A questo punto i cicli F vengono ottenuti come combinazioni dei nodi con $h = F$ individuati in ciascun nodo del ciclo incerto.

Teorema 2.7.1. *Un sistema di rete $\langle N, M_0 \rangle$ è diagnosticabile in riferimento alla classe di guasto T_f^i se e solo se il MBRD non presenta cicli indeterminati relativamente a T_f^i.* ■

Corollario 2.7.1. *Un sistema di rete $\langle N, M_0 \rangle$ è diagnosticabile se e solo se il MBRD non presenta cicli indeterminati rispetto a ogni classe di guasto.* ■

2.7.3 Algoritmo di Johnson per il calcolo dei cicli in un grafo

Abbiamo già visto come la verifica della diagnosticabilità sia basata sui concetti di cicli incerti e indeterminati. Per la ricerca dei cicli incerti all'interno del MBRD si è fatto uso dell'algoritmo di Johnson [23].

Si tratta di un algoritmo che trova tutti i cicli elementari di un grafo orientato. Un *ciclo* è un percorso in cui il primo e l'ultimo nodo sono identici. Un ciclo è *elementare* se nessun nodo appare due volte. L'algoritmo è simile agli algoritmi di Tiernan [34] e Tarjan [33].

Tale algoritmo esclude grafi con loop (archi della forma (v, v)) e archi multipli tra una coppia di vertici.

L'algoritmo di Johnson è fondamentalmente un algoritmo di ricerca in profondità *Depth First Search* (DFS) che è stato modificato per migliorare l'efficienza della ricerca. Le etichette intere $\{1, 2, ..., n\}$ sono assegnate ai nodi del grafo e i cicli sono definiti iniziando dal nodo con etichetta minore. Ogni nodo s del grafo è scelto successivamente, e nel grafo vengono ricercati i cicli legati a quel nodo. Quando i cicli legati al nodo s sono stati trovati, viene scelto il nodo successivo $s+1$ e la ricerca nel grafo continua. L'algoritmo utilizza il concetto di componenti fortemente connesse. Le componenti fortemente connesse hanno la proprietà che tutti i nodi della componente sono mutuamente raggiungibili. Esse vengono utilizzate per individuare le zone del grafo che contengono tutti i cicli che hanno radice in un determinato nodo s. In particolare un grafo F è indotto dal grafo originale G dai nodi $\{s, s+1, ..., n\}$. Le componenti fortemente connesse hanno la proprietà che tutti i nodi sono raggiungibili dagli altri nodi. Allora la componente fortemente connessa contenente il nodo s contiene tutti i cicli che hanno radice in s.

Durante la ricerca per cicli con radice nel nodo s, vengono memorizzate le informazioni relative ai nodi e agli archi che non conducono a cicli. Questo impedisce che questi nodi e archi vengano cercati di nuovo in futuro. Questa informazione è memorizzata usando nodi bloccati e insiemi B. Fondamentalmente, i nodi vengono bloccati quando vengono aggiunti al percorso di ricerca. Ogni nodo ha anche un B-set che contiene i suoi predecessori che dovrebbero venire sbloccati quando il nodo viene sbloccato.

I cicli elementari sono costruiti a partire da un vertice radice s nel sottografo indotto da s. Per evitare che ci siano cicli doppi, un vertice v è bloccato quando viene aggiunto al percorso elementare che inizia in s. Esso rimane bloccato

fino a che ciascun percorso da v a s interseca il percorso elementare corrente in un vertice diverso da s. L'algoritmo accetta un grafo G rappresentato da una struttura di adiacenza A_G composta da una lista di adiacenza $A_G(v)$ per ogni $v \in V$. La lista $A_G(v)$ contiene u se è solo se l'arco $(v, u) \in E$. L'algoritmo assume che i vertici siano rappresentati da numeri interi da 1 a n. L'algoritmo procede con la costruzione dei percorsi elementari a partire dal vertice s. I vertici del percorso elementare corrente sono tenuti in uno stack. Un vertice viene accodato ad un percorso elementare tramite una chiamata alla procedura CIRCUIT e viene cancellato al ritorno da questa chiamata. Quando un vertice v viene aggiunto ad un percorso, esso è bloccato impostando *blocked(v) = true*, così che v non possa essere utilizzato due volte nello stesso percorso. In [26] è riportato l'algoritmo in pseudo-codice ed esempi che spiegano il funzionamento in modo dettagliato.

Capitolo 3

Piattaforma Software

Sommario

In questo capitolo verrà presentata la *Piattaforma Software* realizzata nell'ambito del progetto europeo *Distributed Supervisory Control of Large Plants* (DISC). Il progetto è iniziato a Settembre 2008 e il suo termine è previsto entro l'anno 2011. Alla fine del progetto la *Piattaforma Software* verrà distribuita alle università e alle industrie tramite il *web-server* del progetto [1].

3.1 Descrizione generale della Piattaforma Software

I requisiti del sistema per poter installare la *Piattaforma Software* sono i seguenti:

1. Pentium 4 1 GHz o superiore;
2. 512 Mb ram;
3. Windows Xp o superiori;
4. Windows Installer 3.1 (verrà automaticamemte scaricato ed installato se non è già presente);
5. .Net framework 3.0 (verrà automaticamemte scaricato ed installato se non è già presente).

La *Piattaforma Software* ha lo scopo di integrare, insieme ad altri tool già esistenti, gli algoritmi sviluppati durante il corso del progetto. L'obiettivo che si propone è duplice:

- fornire uno strumento che faciliti il trasferimento di queste tecniche agli utenti finali;
- permettere all'utente di confrontare le diverse metodologie e i diversi tool.

Al fine di garantire un'ampia diffusione della *Piattaforma Software* tra università e industrie occorre che:

- la piattaforma venga rilasciata gratuitamente;
- nella piattaforma sia presente un'interfaccia con strumenti per la modellazione e l'analisi dei SED;
- nella piattaforma sia consentita l'importazione/esportazione di file da/verso diversi tools;
- la piattaforma sia realizzata con un'interfaccia utente uniforme.

Sono state identificate due famiglie di modelli principali come rilevanti per il progetto DISC, gli automi (macchine a stati finiti, catene di Markov) e le reti di Petri (reti etichettate, reti temporizzate sia deterministiche che stocastiche, reti continue, reti ibride). In una prima fase del progetto, sono state raccolte le informazioni circa le esigenze dei gruppi in termini di modelli ad eventi discreti da inserire sulla piattaforma e sui tool software già esistenti da integrare. Le reti di Petri e gli automi sono usati come modello di riferimento per la *Piattaforma Software*.

Il linguaggio per le reti di Petri supportato dalla piattaforma è basato sul *Petri Net Markup Language* (PNML), appartenente alla famiglia XML. Inizialmente, il PNML era destinato ad essere un formato di file per la versione Java della rete di Petri Kernel. Poi si è scoperto che ci sono molti alti gruppi che stanno sviluppando un formato di interscambio tra XML e PNML, quindi probabilmente il PNML è destinato a divenire un formato standard.

E' importante sottolineare che il formato PNML non ha lo scopo di definire tutti i possibili tipi di reti di Petri nel corso di un singolo processo di standardizzazione

(che sarebbe impossibile), ma fornisce linee guida per estendere lo standard in modo da descrivere tipi di reti arbitrarie.

Il cuore della *Piattaforma Software* è basato su un concetto semplice, poichè dovrà supportare diversi tool software (sia Windows che Matlab). Dal momento che quasi ogni strumento basato su Windows usa un diverso formato di file, e anche gli strumenti di base di Matlab usano un formato (.m) che sostanzialmente è un testo ASCII puro, sono stati creati una serie di blocchi funzionali chiamati *plugin* e *adapter* che eseguono il lavoro di conversione dai diversi formati di file.

Un *plugin* è un singolo eseguibile autonomo in grado di effettuare una conversione richiedendo solo pochi parametri, come il nome del file di ingresso e il nome del file di destinazione (e di alcuni parametri extra se non sono presenti nel formato dei file di destinazione). Di solito, nella *Piattaforma Software*, un *plugin* eseguirà una conversione da qualsiasi formato supportato verso il formato Pnml_DISC. Un *adapter* farà il lavoro opposto, ovvero dal formato Pnml_DISC verso altri formati di file. Naturalmente, qualsiasi combinazione di questi *plugin* e/o *adapter* possono essere utilizzati per creare conversioni molto più complicate. Inoltre, poichè ci sono molti tools basati su Matlab, un formato di file chiamato Matlab_DISC agirà come formato di interscambio tra il formato Pnml_DISC e i tools Matlab. In seguito, verranno spiegati più in dettaglio tutti i *plugin* e gli *adapter* creati.

3.2 Descrizione dettagliata della Piattaforma Software

La *Piattaforma Software* è principalmente divisa in quattro aree: *Tools*, *File Conversion*, *PNML Analysis*, *Script Manager*.

3.2.1 Tools

L'area *Tools* permette all'utente di aggiungere/rimuovere tool esterni alla/dalla *Piattaforma Software*. Questo creerà un unico ambiente dove l'utente può vedere e lanciare tutti i tools disponibili. È diviso in tre sotto aree come si può vedere dalla Figura 3.1:

1. *Graphic Tools*;
2. *Matlab Tools*;

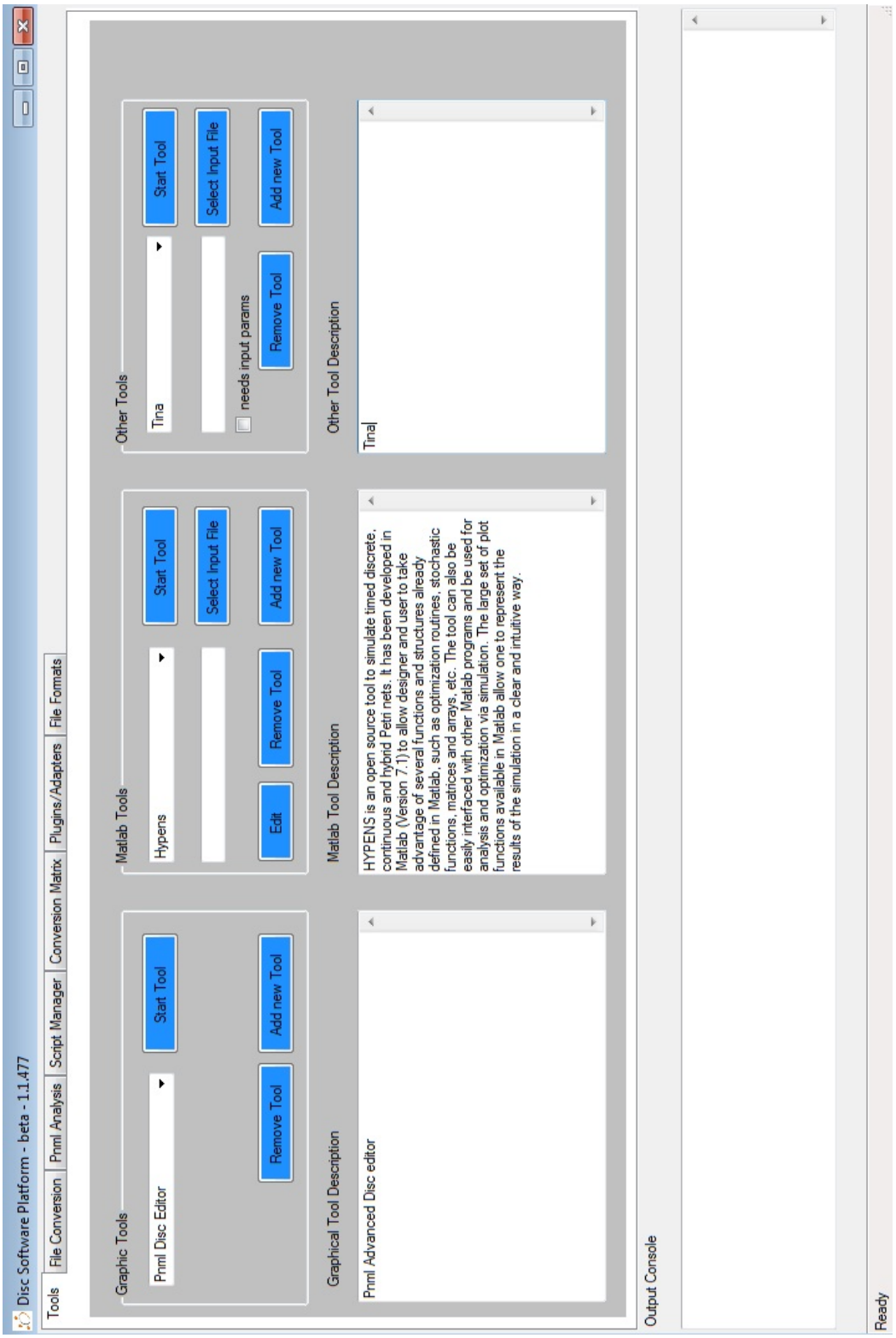

Figura 3.1: Area Tools e le sue sotto-aree: Graphic Tools, Matlab Tools e Other Tools

3. *Other Tools*.

Si noti che, la divisione non ha una rilevanza teorica ma è stata usata solo per separare i differenti tools presenti nella piattaforma da un punto di vista di usabilità.

La *Piattaforma Software* viene distribuita con:

- *Pnml Disc Editor* (nella *Graphic Tools*);
- i toolbox: *Hypens* (richiede un sistema a 32 bit), *PN_Diag* e *PN_DIAG_2* (in *Matlab Tools*).

Inoltre correntemente sono supportati i seguenti tools esterni:

1. Matlab;
2. *Pipe3* (scaricabile gratuitamente da [3]);
3. *Pipe2* (non più scaricabile da [3] ma comunque supportato);
4. *Tina* (scaricabile gratuitamente [5]);
5. *Desuma* (scaricabile gratuitamente in [24]);

La sotto-area *Graphic Tools* si occupa di tutti i tools esterni che generalmente hanno una interfaccia grafica con l'utente e poi possono semplicemente essere usati per lanciare l'interfaccia grafica con l'utente (GUI) e l'interazioni con essa. Per questo lavoro di tesi è stato necessario aggiungere a tale area i seguenti tool: *Pipe2*, *Pipe3* e *Desuma*.

La sotto-area *Matlab Tools* gestisce i tools "Matlab-based" e ha un campo aggiuntivo per i file di input. Questo è necessario perché di solito un tool Matlab ha bisogno di un file di input su cui lavorare e a tal fine può essere selezionato dall'utente. Per poter lanciare un tool Matlab è necessario fornire (almeno per *Hypens*, *Pn_Diag* e *PN_DIAG_2*) un file di input, ovvero un file Matlab contenente le matrici input necessarie. A questo scopo è possibile creare una rete di Petri in diversi modi (formati di file) e usare la sezione *File Conversion* per convertirla al formato di file Matlab_Hypens (per il toolbox *Hypens*) e PN_Diag_Matlab (per i toolbox *PN_Diag* e *PN_DIAG_2*). Altri tool Matlab, come *SimHPN* non necessitano di alcun file di input in quanto hanno un'interfaccia grafica con l'utente

che si avvia automaticamente quando viene lanciato il tool. Inoltre è prevista la realizzazione di un tool Matlab per studi di diagnosi decentralizzata sempre in riferimento alle reti di Petri. In questo lavoro di tesi verranno utilizzati i toolbox *Pn_Diag* e *PN_DIAG_2*.

La sotto-area *Other Tools* è stata creata al fine di integrare i tool che non hanno un'interfaccia grafica con l'utente, per esempio quelli con righe di comando. Questa sezione ha caratteristiche simili alle sotto-aree *Graphic Tools* e *Matlab Tools*. Per esempio un tool a riga di comando può essere aggiunto alla piattaforma in quest'area e lanciato passando i parametri attraverso il file di input. In questo lavoro di tesi è stato necessario aggiungere il tool *Tina*.

3.2.2 File Conversion

L'area *File Conversion* è il luogo dove l'utente può selezionare e convertire un file prodotto da un tool in un altro formato di file compatibile con un diverso tool, come si può vedere in Figura 3.2. In quest'area l'utente può anche editare sia il file sorgente che il file di destinazione. Per convertire un file l'utente può selezionare il file sorgente sul lato sinistro (il formato del file verrà rilevato automaticamente dalla piattaforma nella maggior parte dei casi, in caso contrario, l'utente deve selezionare il tipo di file in ingresso manualmente prima di procedere alla conversione) e poi premere il tasto *convert* e selezionare il formato del file di destinazione. Premendo il pulsante "OK" la conversione del file si avvia automaticamente. Il risultato della conversione è mostrato nell'output console nella parte inferiore della finestra principale e nel "Target File" della casella di testo. L'editor per il file sorgente e di destinazione è anche selezionabile dall'utente.

3.2.3 PNML analysis

L'area *PNML analysis* mostrata in Figura 3.3 è dedicata all'analisi delle proprietà di una Rete di Petri descritta con il formato di interscambio chiamato Pnml_DISC. L'apertura di un file PNML in quest'area dà luogo a due processi:

1. estrae le proprietà che sono esplicitamente presenti nel file PNML come il numero di posti e transizioni della rete di Petri;
2. usa *Tina* per determinare la limitatezza della rete.

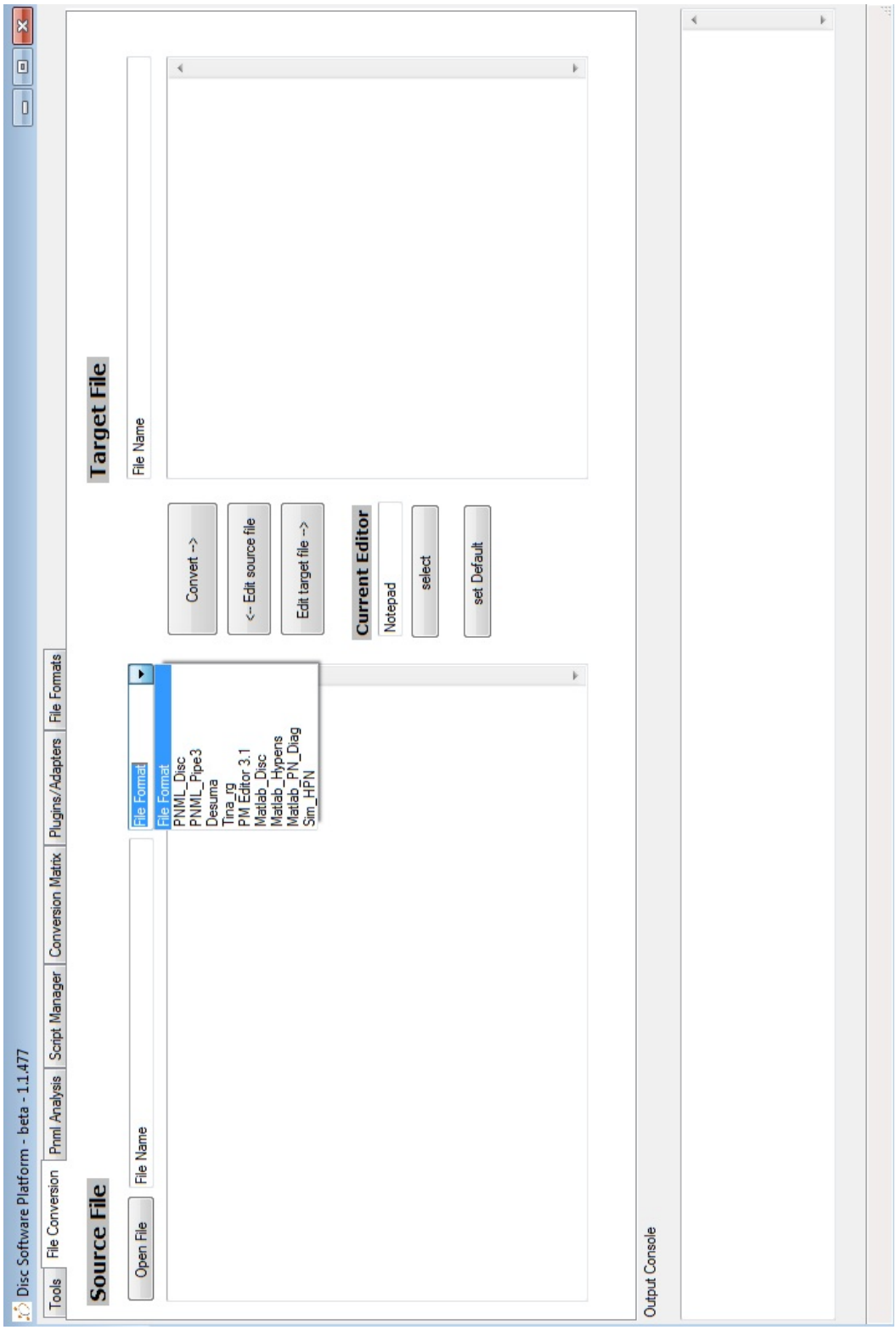

Figura 3.2: Area File Conversion della *Piattaforma Software*

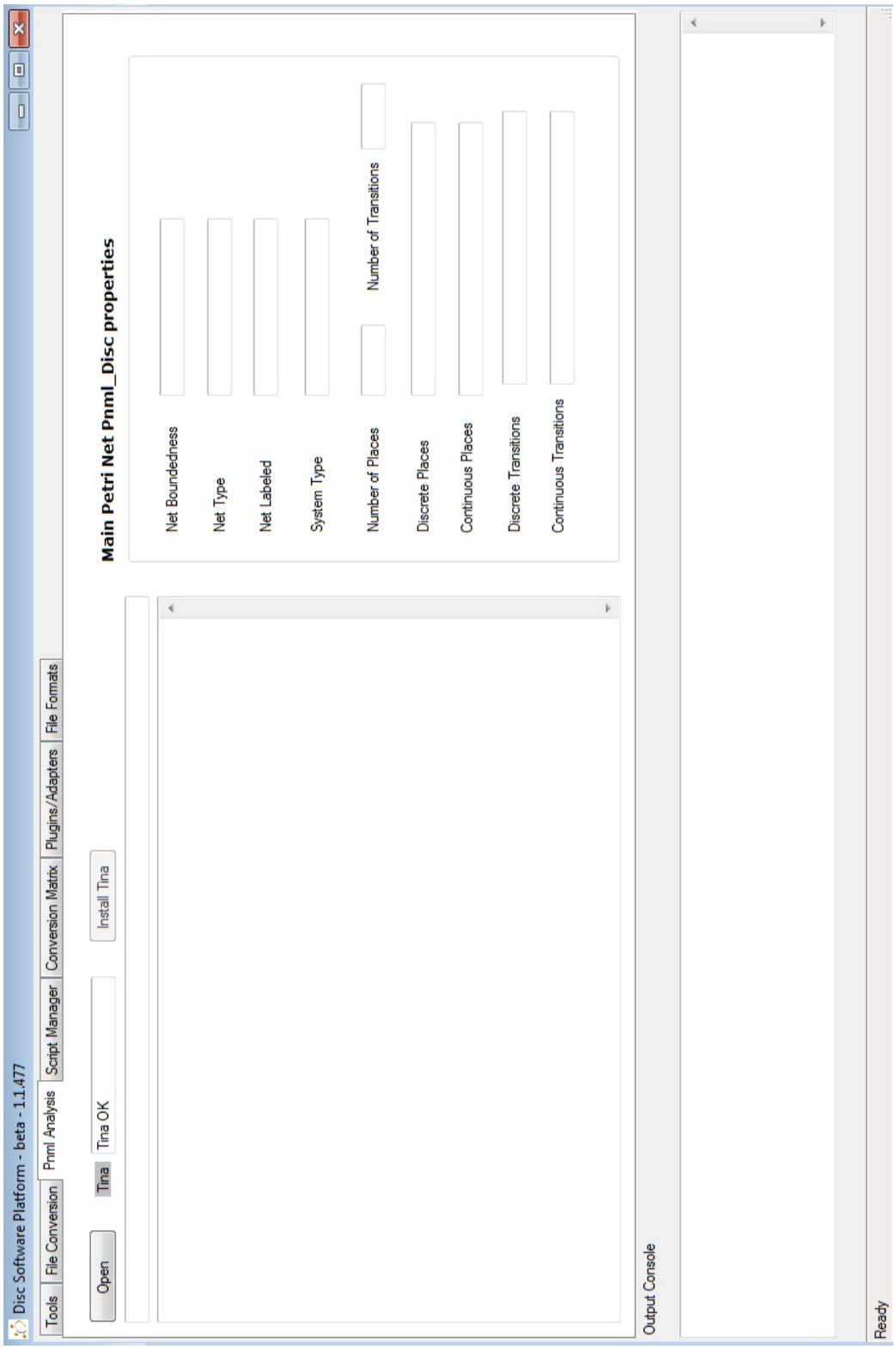

Figura 3.3: Area PNML analysis della *Piattaforma Software*

3.2.4 Script Manager

Nell'area *Script Manager* l'utente può creare e lanciare file di *script* (per esempio file batch, file Matlab, sh scripts). Questa area è molto utile quanto l'utente vuole sfruttare i vantaggi della modularità della *Piattaforma Software*. Tale area si presenta come in Figura 3.4. Per esempio, quando si esegue una conversione del formato di file, la piattaforma combina un numero di *plugin* e/o *adapter* per andare dal formato di file di origine al formato di file di destinazione. Questa sequenza è normalmente incorporata in una matrice di conversione. Ma se l'utente vuole combinare i *plugin* e gli *adapter* in modo personalizzato, può creare uno *script* per risolvere il problema. Inoltre, l'utente può combinare sequenze di conversione e tool lanciando tutto nello stesso *script*, permettendo di arrivare alla soluzione anche di modelli complessi e ripetitivi.

Per poter creare uno *script* occorre cliccare il tasto "new". Una finestra di *popup* chiede il nome del progetto *script*. Ciò significa che verrà creata una cartella per lo *script*. È anche possibile scegliere tra uno script Matlab e un batch. Dopo aver creato la cartellla e lo *script* vuoto è possibile iniziare a digitare il contenuto dello *script*. Come aiuto importante, sul lato destro, sono mostrati una lista dei *plugin* e degli *adapters* della piattaforma. Non è solo un elenco di informazioni, ma è anche possibile trascinare e rilasciare il *plugin/adapter* dalla lista allo *script*. Questa azione permette che i *plugin/adapters* vengano copiati nella cartellla dello *script project* e produce anche la creazione del codice necessario nello *script*. Si è scelto di utlizzare *script* Matlab, di conseguenza la sintassi è in stile Matlab. L'*input file name* e l'*output file name* significano che l'utente dovrà specificare un file in input. A tal proposito, il pulsante "add file" permette all'utente di aggiungere un file in ingresso allo *script project*. Il file è anche copiato nella cartella del progetto. Usando il pulsante "add file" è possibile aggiungere anche diversi tool a riga di comando (come per esempio *Tina*). Dopo aver creato lo *script* è possibile salvare e mandare in esecuzione usando gli appositi tasti.

3.2.5 Altre aree Tab

Le restanti tre aree hanno finalità puramente descrittiva:

1. *Conversion Matrix*: mostra la combinazione di *plugin* e *adapter* necessaria per passare da un formato di file ad un altro, come si può vedere nella Figura 3.5;

2. *Plugins/Adapters*: contiene una lista dei *plugins* e *adapters* disponibili e il percorso delle directory dove essi si trovano;

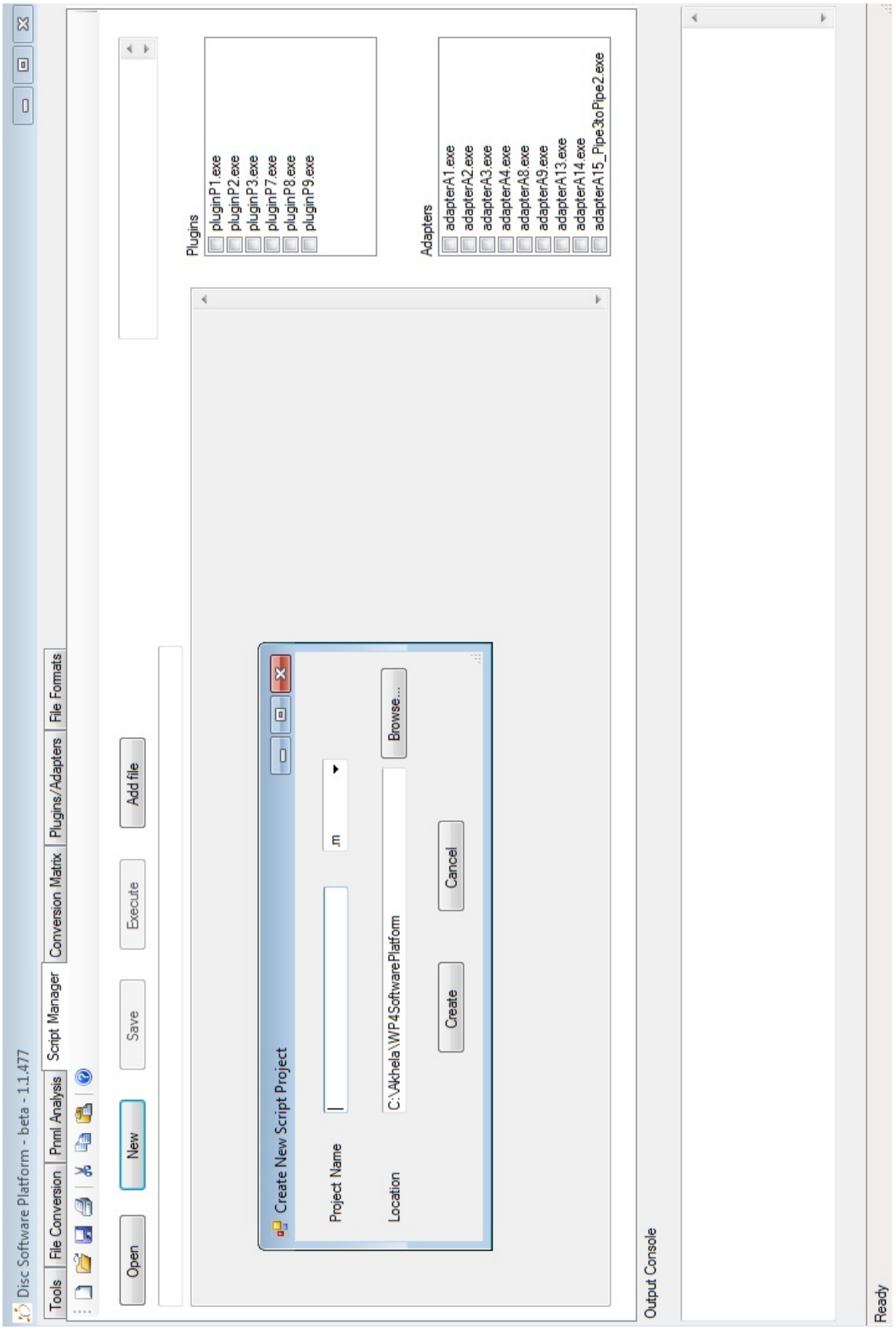

Figura 3.4: Area Script Manager della *Piattaforma Software*

Sigla	Formato del file	Estensione del file
FF1	**Pnml_DISC**	**.pnml**
FF2	**Pipe3_PNML**	**.xml**
FF3	**Desuma**	**.fsm**
FF4	**Tina reachbility graph output**	**.txt**
FF9	**Matlab_DISC**	**.m**
FF10	**Matlab_Hypens**	**.m**
FF11	**Matlab_Pn_Diag**	**.m**
FF12	Matlab_Ddt	.m
FF15	Matlab_HyPnSim	.m
FF16	MinMaxGd	.sci/.sce

Tabella 3.1: Area File Formats della *Piattaforma Software*

3. *File Formats*: spiegano come interpretare la *Conversion Matrix*, riassumiamo questo *tab* nella Tabella 3.1, dove sono scritti in rosso i formati di file presi in considerazione in questo lavoro di tesi.

3.2.6 Plugins e adapters supportati correntemente

In tale sezione elenchiamo i *plugin* e gli *adapters* sviluppati finora.

Iniziamo con i *plugin* che sono riassunti in Tabella 3.2:

- *pluginP1*: converte una rete di Petri descritta attraverso il Pipe3_Pnml nel formato Pnml_DISC, cioè permette di passare da una rete di Petri disegnata con Pipe3 al formato Pnml_DISC, in cui si può visualizzare di quanti posti, transizioni (sia osservabili che non) è composta la rete;
- *pluginP2*: converte una rete di Petri descritta attraverso il formato Matlab_DISC nel formato Pnml_DISC, cioè una volta fornito il file.m in formato Matlab_DISC si può ottenre il formato Pnml_DISC ed avere così un riassunto di tutte le caratteristiche della rete di Petri;
- *pluginP3*: converte un automa descritto attraverso *Desuma*, quindi nel formato di estensione .fsm nel formato Pnml_DISC, cioè possiamo passare dall'automa disegnato con *Desuma* al formato Pnml_DISC supportato dalla piattaforma;

	FF1	**FF2**	**FF3**	**FF4**	**FF10**	**FF11**	**FF9**	**FF12**	**FF15**	**FF16**
FF1	x	A2	A3 or A2+A15+tina+P7	A2+A15+tina	A1+A8	A1+A9	A1	A1+A10	A1+A13	A1+A14
FF2	P1	x	P1+A3 or P1+A2+A15+tina+P7	P1+A2+A15+tina	P1+A1+A8	P1+A1+A9	P1+A1	P1+A1+A10	P1+A1+A13	P1+A1+A14
FF3	P3	P3+A2	x	P3+A2+A15+tina	P3+A1+A8	P3+A1+A9	P3+A1	P3+A1+A10	P3+A1+A13	P3+A1+A14
FF4	P7+P3	P7+P3+A2	P7	x	P7+P3+A1+A8	P7+P3+A1+A9	P7+P3+A1	P7+P3+A1+A10	P7+P3+A1+A13	P7+P3+A1+A14
FF10	P8+P2	P8+P2+A2	P8+P2+A3 or P8+P2+A2+A15+tina+P7	P8+P2+A2+A15+tina	x	P8+A9	P8	P8+A10	P8+A13	P8+A14
FF11	P9+P2	P9+P2+A2	P9+P2+A3 or P9+P2+A2+A15+tina+P7	P9+P2+A2+A15+tina	P9+A8	x	P9	P9+A10	P9+A13	P9+A14
FF9	P2	P2+A2	P2+A3 or P2+A2+A15+tina+P7	P2+A2+A15+tina	A8	A9	x	A10	A13	A14
FF12	P10+P2	P10+P2+A2	P10+P2+A3 or P10+P2+A2+A15+tina+P7	P10+P2+A2+A15+tina	P10+A8	P10+A9	P10	x	P10+A13	P10+A14
FF15	P13+P2	P13+P2+A2	P13+P2+A3 or P13+P2+A2+A15+tina+P7	P13+P2+A2+A15+tina	P13+A8	P13+A9	P13	P13++A10	x	P13+A14
FF16	P14+P2	P14+P2+A2	P14+P2+A3 or P14+P2+A2+A15+tina+P7	P14+P2+A2+A15+tina	P14+A8	P14+A9	P14	P14+A10	P14+A13	x

Figura 3.5: Possibili conversioni della *Piattaforma Software*

pluginP1.exe	da	Pipe3_Pnml	a	Pnml_DISC
pluginP2.exe	da	Matlab_DISC	a	Pnml_DISC
pluginP3.exe	da	Desuma	a	Pnml_DISC
pluginP7.exe	da	Tina Reachability Graph	a	Desuma
pluginP8.exe	da	Hypens_Matlab	a	Matlab_DISC
pluginP9.exe	da	Pn_Diag_Matlab	a	Matlab_DISC
pluginP10.exe	da	Matlab_Ddt	a	Matlab_DISC
pluginP13.exe	da	Matlab_HybPnSim	a	Matlab_DISC
pluginP14.exe	da	MinMaxGd	a	Matlab_DISC

Tabella 3.2: *Plugin* supportati dalla *Piattaforma Software*

- *pluginP7*: converte il grafo di raggiungibilità generato da *Tina* nel formato di estensione .txt in un automa descritto attraverso *Desuma* (.fsm);
- *pluginP8*: permette il passaggio da Hypens_Matlab al formato Matlab_Disc;
- *pluginP9*: converte una rete di Petri descritta attraverso PN_Diag_Matlab in Matlab_DISC, cioè permette di passare dal file.m da mandare in ingresso ai toolbox (per il momento *PN_DIAG* e *PN_DIAG_2*), quindi dal file contenente tutte le informazioni per definire la rete di Petri al formato Matlab_DISC sempre con estensione .m;
- *pluginP10*: è ancora da scrivere, in quanto è necessario per prima cosa realizzare il tool di riferimento, in ogni caso il suo scopo sarà quello di convertire il formato Matlab_Ddt nel formato Matlab_DISC;
- *pluginP13*: è ancora da scrivere, il suo scopo sarà quello di convertire il formato Matlab_HybPnSim nel formato Matlab_DISC;
- *pluginP14*: è ancora da scrivere, il suo scopo sarà quello di convertire il formato MinMaxGd nel formato Matlab_DISC.

Il tutto è riassunto nella Tabella 3.2

Gli *adapters* invece sono riassunti in Tabella 3.3 e sono:

- *adapterA1*: esegue il lavoro opposto del *pluginP2*, quindi permette la conversione dal formato Pnml_DISC al Matlab_DISC;

- *adapterA2*: esegue il lavoro opposto del *pluginP1*, cioè converte una rete di Petri descritta attraverso il formato di file Pnml_DISC nella stessa rete descritta tramite il formato Pnml_Pipe3;
- *adapterA3*: esegue il lavoro opposto del *pluginP3*, cioè converte una rete di Petri descritta attraverso il formato di file Pnml_DISC in un automa descritto attraverso *Desuma* (.fsm) (sotto specifiche condizioni);
- *adapterA4*: converte una rete di Petri descritta attraverso il formato di file Pnml_DISC nella stessa rete descritta attraverso il PM_Editor_3.1 (.rdp);
- *adapterA8*: esegue il lavoro opposto del *pluginP8*, cioè converte una rete di Petri descritta attraverso il formato di file Matlab_DISC nella stessa rete descritta attraverso lo specifico formato di file Hypens_Matlab;
- *adapterA9*: esegue il lavoro opposto del *pluginP9*, cioè converte una rete di Petri descritta attraverso il formato di file Matlab_DISC nella stessa rete descritta attraverso lo specifico formato di file Pn_Diag_Matlab;
- *adapterA10*: è ancora da scrivere, il suo scopo sarà quello di convertire il formato Matlab_DISC nel formato Matlab_Ddt, non appena il tool sarà pronto;
- *adapterA13*: effettua la conversione dal formato Matlab_DISC al formato Matlab_HybPnSim, ovvero quello per il tool *SimHPN*;
- *adapterA14*: effettua la conversione dal formato Matlab_DISC al formato MinMaxGd (.sci/.sce), ovvero quello per il tool *Timed Synchronization Graphs*;
- *adapterA15_Pipe3toPipe2*: effettua la conversione da una rete di Petri disegnata con il tool *Pipe3* ad una rete disegnata con il tool *Pipe2*, quindi non si ha una conversione di estensione del file, in quanto sono entrambi in .xml. Nota che per problemi di visualizzazione il nome riportato in Tabella 3.3 è solo *adapterA15*.

adapterA1.exe	da	Pnml_DISC	a	Matlab_DISC
adapterA2.exe	da	Pnml_DISC	a	Pipe3_Pnml
adapterA3.exe	da	Pnml_DISC	a	Desuma
adapterA4.exe	da	Pnml_DISC	a	PM_Editor_3.1
adapterA8.exe	da	Matlab_DISC	a	Matlab_Hypens
adapterA9.exe	da	Matlab_DISC	a	Pn_Diag_Matlab
adapterA10.exe	da	Matlab_DISC	a	Matlab_Ddt
adapterA13.exe	da	Matlab_DISC	a	Matlab_HybPnSim
adapterA14.exe	da	Matlab_DISC	a	MinMaxGd
adapterA15.exe	da	Pipe3	a	Pipe2

Tabella 3.3: *Adapters* supportati dalla *Piattaforma Software*

Capitolo 4

Software testati

Sommario

In questo capitolo verranno richiamati brevemente i software testati all'interno della tesi. In particolare, daremo una descrizione del software *PN_DIAG* di Pocci [28] e il software *PN_DIAG_2* di Perria [26], entrambi riguardanti il problema della diagnosticabilità delle reti di Petri, e realizzati dal gruppo di automatica dell'università di Cagliari. Inoltre verrà richiamato il software UMDES *Discrete Event Systems Modeled By Finite State Machine*, sviluppato dal gruppo *Discrete Event Systems* dell'università del Michigan sotto la coordinazione dei Professori Lafortune e Teneketzis. Quest'ultimo software inserito all'interno del tool *Desuma*, contiene comandi per la definizione di una macchina a stati finiti, per la diagnosi dei guasti e per il controllo supervisivo. La parte interessante per questo lavoro è quella riguardante la diagnosi dei guasti. Tutti questi software sono stati testati sia all'interno che all'esterno della *Piattaforma Software* (vedi Capitolo 3).

4.1 Software PN_DIAG

Il software *PN_DIAG* realizzato da Pocci [28] racchiude l'approccio alla diagnosi dei *sistemi ad eventi discreti* proposto da Cabasino *et al.* in [13], basato sulle reti di Petri etichettate. La procedura proposta è basata su precedenti risultati sulle RdP non etichettate e ci permette di considerare, in aggiunta, quegli eventi che sono indistinguibili, ossia eventi che producono un segnale di uscita che è osservabile, ma che può essere comune ad altri. Il loro approccio è basato

sulla nozione di marcatura di base e di vettori di giustificazione ed è mostrato come, nel caso delle RdP limitate, la parte gravosa della procedura possa essere effettuata *off-line*, a partire dal *Basis Reachability Graph* (BRG), che potrà essere usato come diagnosticatore. Sempre in questo software è stato preso in considerazione il problema della diagnosticabilità per RdP limitate, presentato da Cabasino *et al.* in [13]. La diagnosticabilità è basata sull'analisi di due grafi che dipendono dalla struttura della rete, incluso il modello di guasto e la marcatura iniziale. Il primo grafo è chiamato *Modified Basis Reachability Graph* (MBRG), il secondo *Basis Reachability Diagnoser* (BRD). Il BRG, il MBRG e il BRD sono stati spiegati, insieme alle condizioni necessarie e sufficienti per la diagnosticabilità nel Capitolo 2. La memoria necessaria per il calcolo della diagnosticabilità risulterà maggiore di quella richiesta per la diagnosi. Infatti, in questo caso sarà necessario tenere conto non solo delle marcature di base ma anche di tutte quelle marcature raggiungibili con lo scatto di transizioni di guasto.

Il toolbox realizzato da Pocci è stato inserito all'interno della *Piattaforma Software* sotto il nome di *PN_DIAG*. Una volta fornito il file di input, contenente le matrici Pre e $Post$, il vettore delle marcature iniziali M_0, il vettore di celle F contenente le transizioni di guasto, il vettore di celle L contenente le transizioni (osservabili, non osservabili e/o di guasto) e il vettore di celle E contenente le etichette associate alle transizioni, alla piattaforma, questa esegue il file.m chiamato: main_PN_DIAG.m, riportato in Appendice A.1.5, che restituisce il grafo di raggiungibilità, il BRG, il MBRG, il BRD e ci fornisce le informazioni sulla diagnosticabilità o meno del sistema dato in ingresso.

4.2 Software PN_DIAG_2

Il software *PN_DIAG_2* realizzato da Perria [26] si occupa delle condizioni necessarie e sufficienti per la diagnosticabilità di RdP limitate e fornisce un metodo sistematico per l'analisi della diagnosticabilità. Per questo motivo, è stata necessaria la costruzione di due grafi etichettati e orientati denominati rispettivamente: *Modified Basis Reachability Graph* (MBRG) e *Modified Basis Reachability Diagnoser* (MBRD). Il MBRG trattato nella tesi di Pocci [28] è una variante del BRG, e il MBRD è una variante del BRD. In breve, questo software consiste nel determinare se nel MBRD sono presenti dei cicli incerti e nel verificare se sono soddisfatte determinate condizioni su di essi. Anche per questo software è stato assunto che l'insieme delle transizioni possa essere diviso in $T = T_o \cup T_u$. L'insieme di transizioni non osservabili è suddiviso a sua volta in due sottoinsiemi, detti $T_u = T_f \cup T_{reg}$, dove T_f include tutte le transizioni di guasto mentre T_{reg}

include tutte le transizioni relative a eventi non osservabili ma regolari. L'insieme T_f è ulteriormente suddiviso in r sottoinsiemi T_f^*, dove $i = 1,...,r$, che modellano le differenti classi di guasto.

Il toolbox realizzato da Perria è stato inserito all'interno della *Piattaforma Software* sotto il nome di *PN_DIAG_2*. Una volta fornito il file di input, contenente il MBRG (calcolato tramite il software *PN_DIAG*), le matrici Pre e $Post$, il vettore della marcatura iniziale M_0, il vettore di celle F contenente le transizioni di guasto, il vettore di celle L contenente le transizioni (osservabili, non osservabili e/o di guasto) e il vettore di celle E contenente le etichette associate alle transizioni, alla piattaforma, questa esegue il file.m chiamato: launch_DISC.m, riportato in Appendice A.1.6 che restituisce il MBRG, il MBRD (al cui interno è contenuto l'algoritmo di Johnson) e ci fornisce le informazioni sulla diagnosticabilità o meno del sistema dato in ingresso.

4.3 Toolbox DESUMA

Questo toolbox contiene le librerie UMDES.

UMDES contiene comandi per la definizione di una macchina a stati finiti (circa 30 comandi), per la diagnosi dei guasti (circa 10 comandi) e per il controllo supervisivo (circa 10 comandi). Per quanto riguarda la parte riguardante la diagnosi dei guasti vi sono comandi che consentono la costruzione del modello, la costruzione dei diagnosticatori e la verifica della diagnosticabilità (cicli indeterminati). Il toolbox *Desuma* e di conseguenza la libreria dei comandi UMDES è disponibile sul *web-server* [24]. Per questo lavoro di tesi, che mira a mettere a confronto i toolbox di Pocci [28] e Perria [26] con *Desuma* (integrazione delle librerie UMDES), sono stati usati solo alcuni comandi:

- *create_fsm*: questo programma guida l'utente verso la creazione di un automa a stati finiti, dato un qualsiasi ingresso genera in uscita un file con estensione .fsm, questo comando è stato usato per lo più per generare il file con estensione .ft, necessario per contenere gli eventi di guasto;
- *diag_UR*: questo programma genera un diagnosticatore con guasti multipli per il sistema dato, in ingresso viene fornito l'automa a stati finiti (file.fsm) e la partizione dei guasti (file.ft), genera in uscita il diagnosticatore del sistema in ingresso e lo scrive in un opportuno file d'uscita;
- *dcycle*: questo programma verifica la diagnosticabilità del sistema. Infatti, dati in ingresso l'automa a stati finiti (file.fsm) e la partizione dei guasti

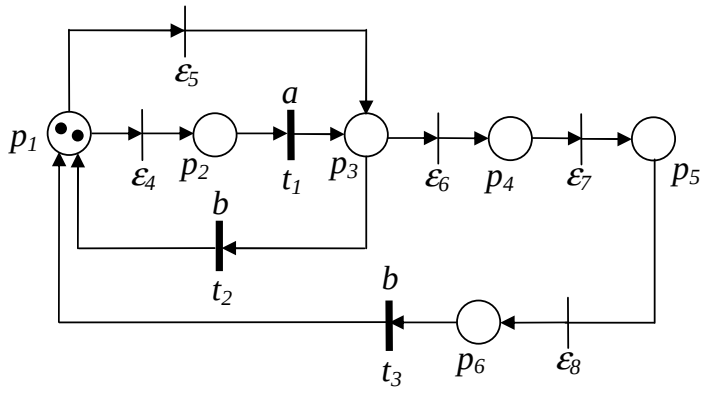

Figura 4.1: Esempio di rete di Petri con un'unica classe di guasto

(file.ft), trova i cicli F_i-indeterminati nel diagnosticatore e li scrive in un opportuno file d'uscita.

4.4 Esempi di funzionamento dei toolbox

In questa sezione vengono presentati alcuni esempi di funzionamento corretto dei toolbox *PN_DIAG* e *PN_DIAG_2* per la diagnosticabilità di RdP, per quanto riguarda invece la diagnosticabilità degli automi a stati finiti da esse ricavati si rimanda al Capitolo 6.

4.4.1 Esempio 1

Consideriamo la Rete di Petri in Figura 4.1, in cui l'insieme delle transizioni osservabili è $T_o = \{t_1, t_2, t_3\}$, l'insieme delle transizioni non osservabili è $T_u = \{\varepsilon_4, \varepsilon_5, \varepsilon_6, \varepsilon_7, \varepsilon_8\}$. L'unica classe di guasto è $T_f = \{\varepsilon_5, \varepsilon_7\}$. La funzione di etichettatura è tale che: $\mathscr{L}(t_1) = a$, $\mathscr{L}(t_2) = \mathscr{L}(t_3) = b$.

Il toolbox *PN_DIAG* calcola il grafo di raggiungibilità, il BRG, il MBRG, il BRD e verifica la diagnosticabilità analizzando la presenza di cicli indeterminati. In Appendice A.1.1 è riportato il file esempio1_Pocci.m nell' che mandiamo in esecuzione per ottenere informazioni sulla diagnosticabilità della rete in Figura 4.1. Tali informazioni vengono riportate sul *Command Windows* di Matlab al termine dell'esecuzione. Grazie alla chiamata delle funzioni:

- *showBRG(brg)*;

- *showMBRG(mbrg)*;
- *showMBRD(brd)*;

possiamo visualizzare i grafi relativi alla Rete di Petri in Figura 4.1.

Riportiamo qui di seguito le informazione restituite sul *Command Windows* di Matlab.

BRG: Basis Reachability Graph

```
Basis Reachability Graph node´s number n = 3

# Marking M0=[2 0 0 0 0 0]´
  x=[1]

  Observable transitions enabled to fire:
  a(t1) -> M1: e=[1 0 0 0 0]
  b(t2) -> M0: e=[0 1 0 0 0]
  b(t3) -> M0: e=[0 1 1 1 1]

        *******

# Marking M1=[1 0 1 0 0 0]´
  x=[1]

  Observable transitions enabled to fire:
  a(t1) -> M2: e=[1 0 0 0 0]
  b(t2) -> M0: e=[0 0 0 0 0]
  b(t3) -> M0: e=[0 0 1 1 1]

        *******

# Marking M2=[0 0 2 0 0 0]´
  x=[1]

  Observable transitions enabled to fire:
  b(t2) -> M1: e=[0 0 0 0 0]
  b(t3) -> M1: e=[0 0 1 1 1]

        *******
```

MBRG: Modified Basis Reachability Graph

```
Modified Basis Reachability Graph node´s number n = 6

# Marking M0=[2 0 0 0 0 0]´
  x=[1]

  Observable and faulty transitions enabled to fire:
  a(t1) -> M1: e=[1 0 0]
  eps5  -> M1: e=[0 0 0]

        *******
```

```
# Marking M1=[1 0 1 0 0 0]´
  x=[1]

  Observable and faulty transitions enabled to fire:
  a(t1) -> M2: e=[1 0 0]
  b(t2) -> M0: e=[0 0 0]
  eps5  -> M2: e=[0 0 0]
  eps7  -> M3: e=[0 1 0]

        *******

# Marking M2=[0 0 2 0 0 0]´
  x=[1]

  Observable and faulty transitions enabled to fire:
  b(t2) -> M1: e=[0 0 0]
  eps7  -> M4: e=[0 1 0]

        *******

# Marking M3=[1 0 0 0 1 0]´
  x=[1]

  Observable and faulty transitions enabled to fire:
  a(t1) -> M4: e=[1 0 0]
  b(t3) -> M0: e=[0 0 1]
  eps5  -> M4: e=[0 0 0]

        *******

# Marking M4=[0 0 1 0 1 0]´
  x=[1]

  Observable and faulty transitions enabled to fire:
  b(t2) -> M3: e=[0 0 0]
  b(t3) -> M1: e=[0 0 1]
  eps7  -> M5: e=[0 1 0]

        *******

# Marking M5=[0 0 0 0 2 0]´
  x=[0]

  Observable and faulty transitions enabled to fire:
  b(t3) -> M3: e=[0 0 1]

        *******
```

BRD: Basis Reachability Diagnoser

```
Basis Reachability Diagnoser node´s number N = 9

# Node N0    Delta=[1]

  Basis Markings and corresponding X-fault and H:
  Marking M0=[2 0 0 0 0 0]´    x=[1]    h=[N]

  Observable transitions enabled to fire:
  a  -> N1
```

```
  b  -> N2

        *******

# Node N1    Delta=[1]

  Basis Markings and corresponding X-fault and H:
  Marking M1=[1 0 1 0 0 0]´    x=[1]    h=[N]

  Observable transitions enabled to fire:
  a  -> N3
  b  -> N4

        *******

# Node N2    Delta=[3]

  Basis Markings and corresponding X-fault and H:
  Marking M0=[2 0 0 0 0 0]´    x=[1]    h=[F]

  Observable transitions enabled to fire:
  a  -> N5
  b  -> N2

        *******

# Node N3    Delta=[1]

  Basis Markings and corresponding X-fault and H:
  Marking M2=[0 0 2 0 0 0]´    x=[1]    h=[N]

  Observable transitions enabled to fire:
  b  -> N6

        *******

# Node N4    Delta=[2]

  Basis Markings and corresponding X-fault and H:
  Marking M0=[2 0 0 0 0 0]´    x=[1]    h=[N]
  Marking M0=[2 0 0 0 0 0]´    x=[1]    h=[F]

  Observable transitions enabled to fire:
  a  -> N6
  b  -> N2

        *******

# Node N5    Delta=[3]

  Basis Markings and corresponding X-fault and H:
  Marking M1=[1 0 1 0 0 0]´    x=[1]    h=[F]

  Observable transitions enabled to fire:
  a  -> N7
  b  -> N2

        *******

# Node N6    Delta=[2]

  Basis Markings and corresponding X-fault and H:
```

```
  Marking M1=[1 0 1 0 0 0]´    x=[1]    h=[N]
  Marking M1=[1 0 1 0 0 0]´    x=[1]    h=[F]

  Observable transitions enabled to fire:
  a  -> N8
  b  -> N4

        *******

# Node N7    Delta=[3]

  Basis Markings and corresponding X-fault and H:
  Marking M2=[0 0 2 0 0 0]´    x=[1]    h=[F]

  Observable transitions enabled to fire:
  b  -> N5

        *******

# Node N8    Delta=[2]

  Basis Markings and corresponding X-fault and H:
  Marking M2=[0 0 2 0 0 0]´    x=[1]    h=[N]
  Marking M2=[0 0 2 0 0 0]´    x=[1]    h=[F]

  Observable transitions enabled to fire:
  b  -> N6

        *******
```

Riportiamo i risultati ottenuti nella Tabella 4.1, strutturata nel seguente modo:

- la colonna 1 indica il numero di gettoni della marcatura iniziale, x;
- le colonne 2 e 3 indicano la cardinalità del grafo di raggiungibilità ($|R|$) e il tempo di calcolo espresso in secondi (t_R), rispettivamente;
- le colonne 4 e 5 indicano la cardinalità del BRG ($|BRG|$) e il tempo di calcolo espresso in secondi ($t_{BRG}[s]$), rispettivamente;
- le colonne 6 e 7 indicano la cardinalità del MBRG ($|MBRG|$) e il tempo di calcolo espresso in secondi ($t_{MBRG}[s]$), rispettivamente;
- le colonne 8 e 9 indicano la cardinalità del BRD ($|BRD|$) e il tempo di calcolo espresso in secondi ($t_{BRD}[s]$), rispettivamente;
- la colonna 10 indica il tempo di calcolo della diagnosticabilità espresso in secondi ($t_{diag}[s]$);
- la colonna 11 indica se il sistema è diagnosticabile oppure no.

PN_DIAG										
x	$\|R\|$	$t_R[s]$	$\|BRG\|$	$t_{BRG}[s]$	$\|MBRG\|$	$t_{MBRG}[s]$	$\|BRD\|$	$t_{BRD}[s]$	$t_{diag}[s]$	$Diag?$
2	21	0.0582	3	0.1606	6	0.0963	9	0.2469	0.6077	no

Tabella 4.1: Risultati della simulazione effettuata con il toolbox *PN_DIAG* prendendo in considerazione la rete in Figura 4.1

Il toolbox *PN_DIAG_2* calcola il MBRG, il MBRD e verifica la diagnosticabilità analizzando la presenza di cicli indeterminati. In Appendice A.1.2 è riportato il file esempio1_Perria.m che mandiamo in esecuzione per ottenere informazioni sulla diagnosticabilità della rete in Figura 4.1. Tali informazioni vengono riportate sul *Command Windows* di Matlab al termine dell'esecuzione. Grazie alla chiamata delle funzioni:

- *showMBRG(T)*;
- *showMBRD(D,numTclas)*, dove D è il MBRD rappresentato da una matrice di celle, e $numTclas$ è il numero di classi di guasto;

possiamo visualizzare i grafi relativi alla Rete di Petri in Figura 4.1.

Riportiamo solo il *Command Windows* di Matlab restituito per il MBRD.

MBRD: Modified Basis Reachability Graph

```
Il numero di nodi del Modified Basis Reachability Diagnoser è: 9

 # Nodo N1  Delta = [2]
 Le marcature appartenenti al nodo e i corrispondenti vettori h sono:
 Marcatura M0 = [2 0 0 0 0 0]    h = [N]
 Marcatura M1 = [1 0 1 0 0 0]    h = [F]
 Marcatura M2 = [0 0 2 0 0 0]    h = [F]
 Marcatura M3 = [1 0 0 0 1 0]    h = [F]
 Marcatura M4 = [0 0 1 0 1 0]    h = [F]
 Marcatura M5 = [0 0 0 0 2 0]    h = [F]

 Transizioni osservabili abilitate a scattare:
 a ---> N2
 b ---> N3

                  *********

 # Nodo N2  Delta = [2]
 Le marcature appartenenti al nodo e i corrispondenti vettori h sono:
 Marcatura M1 = [1 0 1 0 0 0]    h = [N]
 Marcatura M2 = [0 0 2 0 0 0]    h = [F]
 Marcatura M4 = [0 0 1 0 1 0]    h = [F]
 Marcatura M3 = [1 0 0 0 1 0]    h = [F]
```

```
Marcatura M5 = [0 0 0 0 2 0]    h = [F]

Transizioni osservabili abilitate a scattare:
a ---> N4
b ---> N5

                  *********

# Nodo N3  Delta = [3]
Le marcature appartenenti al nodo e i corrispondenti vettori h sono:
Marcatura M0 = [2 0 0 0 0 0]    h = [F]
Marcatura M1 = [1 0 1 0 0 0]    h = [F]
Marcatura M3 = [1 0 0 0 1 0]    h = [F]
Marcatura M2 = [0 0 2 0 0 0]    h = [F]
Marcatura M4 = [0 0 1 0 1 0]    h = [F]
Marcatura M5 = [0 0 0 0 2 0]    h = [F]

Transizioni osservabili abilitate a scattare:
a ---> N6
b ---> N3

                  *********

# Nodo N4  Delta = [2]
Le marcature appartenenti al nodo e i corrispondenti vettori h sono:
Marcatura M2 = [0 0 2 0 0 0]    h = [N]
Marcatura M4 = [0 0 1 0 1 0]    h = [F]
Marcatura M5 = [0 0 0 0 2 0]    h = [F]

Transizioni osservabili abilitate a scattare:
b ---> N7

                  *********

# Nodo N5  Delta = [2]
Le marcature appartenenti al nodo e i corrispondenti vettori h sono:
Marcatura M0 = [2 0 0 0 0 0]    h = [N]
Marcatura M1 = [1 0 1 0 0 0]    h = [F]
Marcatura M3 = [1 0 0 0 1 0]    h = [F]
Marcatura M0 = [2 0 0 0 0 0]    h = [F]
Marcatura M2 = [0 0 2 0 0 0]    h = [F]
Marcatura M4 = [0 0 1 0 1 0]    h = [F]
Marcatura M5 = [0 0 0 0 2 0]    h = [F]

Transizioni osservabili abilitate a scattare:
a ---> N7
b ---> N3

                  *********

# Nodo N6  Delta = [3]
Le marcature appartenenti al nodo e i corrispondenti vettori h sono:
Marcatura M1 = [1 0 1 0 0 0]    h = [F]
Marcatura M2 = [0 0 2 0 0 0]    h = [F]
Marcatura M4 = [0 0 1 0 1 0]    h = [F]
Marcatura M3 = [1 0 0 0 1 0]    h = [F]
Marcatura M5 = [0 0 0 0 2 0]    h = [F]

Transizioni osservabili abilitate a scattare:
a ---> N8
b ---> N3
```

```
                    *********

# Nodo N7  Delta = [2]
Le marcature appartenenti al nodo e i corrispondenti vettori h sono:
Marcatura M1 = [1 0 1 0 0 0]    h = [N]
Marcatura M3 = [1 0 0 0 1 0]    h = [F]
Marcatura M1 = [1 0 1 0 0 0]    h = [F]
Marcatura M2 = [0 0 2 0 0 0]    h = [F]
Marcatura M4 = [0 0 1 0 1 0]    h = [F]
Marcatura M5 = [0 0 0 0 2 0]    h = [F]

Transizioni osservabili abilitate a scattare:
a ---> N9
b ---> N5

                    *********

# Nodo N8  Delta = [3]
Le marcature appartenenti al nodo e i corrispondenti vettori h sono:
Marcatura M2 = [0 0 2 0 0 0]    h = [F]
Marcatura M4 = [0 0 1 0 1 0]    h = [F]
Marcatura M5 = [0 0 0 0 2 0]    h = [F]

Transizioni osservabili abilitate a scattare:
b ---> N6

                    *********

# Nodo N9  Delta = [2]
Le marcature appartenenti al nodo e i corrispondenti vettori h sono:
Marcatura M2 = [0 0 2 0 0 0]    h = [N]
Marcatura M4 = [0 0 1 0 1 0]    h = [F]
Marcatura M2 = [0 0 2 0 0 0]    h = [F]
Marcatura M5 = [0 0 0 0 2 0]    h = [F]

Transizioni osservabili abilitate a scattare:
b ---> N7

                    *********
```

I risultati ottenuti sono riportati nella Tabella 4.2, strutturata nel seguente modo:

- la colonna 1 indica il numero di gettoni della marcatura iniziale, x;
- le colonne 2 e 3 indicano la cardinalità dell'MBRG ($|MBRG|$) e il tempo di calcolo espresso in secondi ($t_{MBRG}[s]$), rispettivamente;
- le colonne 4 e 5 indicano la cardinalità dell'MBRD ($|MBRD|$) e il tempo di calcolo espresso in secondi ($t_{MBRD}[s]$), rispettivamente;
- la colonna 6 indica il numero di cicli incerti trovati all'interno dell'MBRD ($\#\ c.\ incerti$);
- la colonna 7 indica il tempo di calcolo della diagnosticabilità espresso in secondi ($t_{diag}[s]$);

PN_DIAG_2							
x	$\|MBRG\|$	$t_{MBRG}[s]$	$\|MBRD\|$	$t_{MBRD}[s]$	$\#\ c.\ incerti$	$t_{diag}[s]$	$Diag?$
2	6	0.2445	9	0.5416	2	1.0756	no

Tabella 4.2: Risultati della simulazione effettuata con il toolbox *PN_DIAG_2* prendendo in considerazione la rete in Figura 4.1

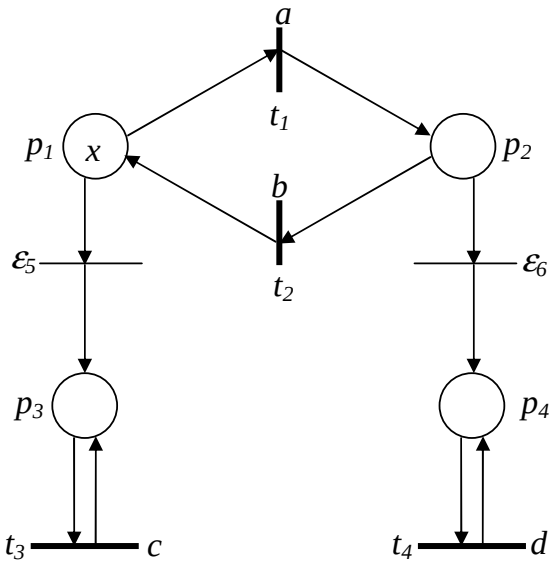

Figura 4.2: Esempio di rete di Petri con due classi di guasto

- la colonna 8 indica se il sistema è diagnosticabile oppure no.

4.4.2 Esempio 2

Consideriamo la rete in Figura 4.2 in cui l'insieme delle transizioni osservabili è $T_o = \{t_1, t_2, t_3, t_4\}$, l'insieme delle transizioni non osservabili è $T_u = \{\varepsilon_5, \varepsilon_6\}$. Sono presenti due classi di guasto $T_f = T_f^1 \cup T_f^2 = \{\varepsilon_5\} \cup \{\varepsilon_6\}$. La funzione di etichettatura è tale che: $\mathscr{L}(t_1) = a$, $\mathscr{L}(t_2) = b$, $\mathscr{L}(t_3) = c$ e $\mathscr{L}(t_4) = d$.

Per prima cosa abbiamo testato il toolbox *PN_DIAG*, che calcola il grafo di raggiungibilità, il BRG, il MBRG, il BRD e verifica la diagnosticabilità analizzando la presenza di cicli indeterminati. In Appendice A.1.3 è riportato il file esempio2_Pocci.m che mandiamo in esecuzione per ottenere informazioni sulla

PN_DIAG										
x	$\|R\|$	$t_R[s]$	$\|BRG\|$	$t_{BRG}[s]$	$\|MBRG\|$	$t_{MBRG}[s]$	$\|BRD\|$	$t_{BRD}[s]$	$t_{diag}[s]$	$Diag?$
1	4	0.0534	4	0.1635	4	0.0512	5	0.2455	0.3709	si
2	10	0.0022	8	0.0393	10	0.0479	9	0.0585	0.2735	no
3	20	0.0071	12	0.0708	20	0.1127	13	0.1043	0.2825	no
4	35	0.0160	16	0.1070	35	0.2356	17	0.1562	0.4997	no
5	56	0.0381	20	0.1655	56	0.4464	21	0.2326	0.8636	no
6	84	0.0833	24	0.2454	84	0.8273	25	0.3349	1.4573	no

Tabella 4.3: Risultati della simulazione effettuata con il toolbox *PN_DIAG* prendendo in considerazione la rete in Figura 4.2

diagnosticabilità della rete in Figura 4.2. Tali informazioni vengono riportate sul *Command Windows* di Matlab al termine dell'esecuzione. I risultati ottenuti sono riportati nella Tabella 4.3, strutturata come la Tabella 4.2. Il *Command Windows* di Matlab restituisce informazioni circa la diagnosticabilità del sistema per per ogni x (quindi per $x = 1,2,3,4,5,6$) a seconda che siano presenti o meno cicli indeterminati.

Abbiamo poi testato il toolbox *PN_DIAG_2*, che calcola il MBRG, il MBRD e verifica la diagnosticabilità analizzando la presenza di cicli indeterminati. In Appendice A.1.4 è riportato il file esempio2_Perria.m che mandiamo in esecuzione per ottenere informazioni sulla diagnosticabilità della rete in Figura 4.2. Tali informazioni vengono riportate sul *Command Windows* di Matlab al termine dell'esecuzione. I risultati ottenuti sono riportati nella Tabella 4.4, strutturata come la Tabella 4.2, con la differenza che ora la colonna $\#\ c.\ incerti$ si divide in due colonne, rispettivamente $classe\ guasto$ (in questo esempio ho classe di guasto una e classe di guasto due) e $\#\ c.\ incerti$ (ovvero il numero di cicli incerti in riferimento ad ogni classe di guasto considerata). Anche con questo toolbox il *Command Windows* di Matlab restituisce informazioni sulla diagnosticabilità della rete al variare del numero dei gettoni, inoltre restituisce anche il numero di cicli incerti per ogni classe di guasto del sistema. Quindi se ad esempio, trova per la classe di guasto 1 un ciclo incerto e per la classe di guasto 2 nessun ciclo incerto, e ci dice che il sistema non è diagnosticabile, vuol dire che quell'unico ciclo incerto è anche indeterminato.

PN_DIAG_2								
x	$\|MBRG\|$	$t_{MBRG}[s]$	$\|MBRD\|$	$t_{MBRD}[s]$	*classe guasto*	*# c. incerti*	$t_{diag}[s]$	*Diag?*
1	4	0.1815	4	0.1112	una due	0 0	0.4384	si
2	10	0.0516	8	0.0802	una due	2 2	0.2402	no
3	20	0.1149	12	0.2541	una due	5 5	0.4211	no
4	35	0.2370	16	0.8970	una due	8 8	1.1605	no
5	56	0.4808	20	3.3092	una due	11 11	3.8484	no
6	84	0.8242	24	12.8923	una due	14 14	13.7765	no

Tabella 4.4: Risultati della simulazione effettuata con il toolbox *PN_DIAG_2* prendendo in considerazione la rete in Figura 4.2

Capitolo 5

Test Piattaforma Software

Sommario

In questo capitolo verranno illustrati i problemi riscontrati durante la fase di test della piattforma. In particolare, sono stati trovati dei bachi all'interno dei *plugin* e degli *adapter* che impedivano il funzionamento corretto dei toolbox inseriti all'interno della *Piattaforma Software*. Inoltre, poichè lo scopo della tesi è confrontare i risultati forniti dai software *PN_DIAG* e *PN_DIAG_2* con quelli forniti da *Desuma*, è stato testato anche quest'ultimo toolbox, ed è stato trovato un baco all'interno del file eseguibile *dcycle*.

5.1 Introduzione all'uso della Piattaforma Software

In questa sezione viene illustrato lo scopo per cui si intende utilizzare la *Piattaforma Software* ma anche dei tool contenuti al suo interno. Quello che si vuole fare è studiare la diagnosticabilità di una Rete di Petri, in qualsiasi formato essa si presenti. Poichè grazie alla piattaforma è possibile passare da un modello reti di Petri a un modello automa, possiamo trarre delle conclusione su quale approccio sia più efficiente. Quindi se per la verifica della diagnosticabilità di un dato sistema sia meglio utilizzare i toolbox realizzati per le reti di Petri (*PN_DIAG* e *PN_DIAG_2*) oppure sia migliore utilizzare il tool *Desuma* realizzato per gli automi, in particolare facendo uso degli eseguibili *diag_UR* e *dcycle*, che fanno parte della libreria UMDES. La *Piattaforma Software* grazie ai convertitori realizzati permette entrambi gli studi, in particolare grazie ad essa è possibile

mettere a confronto in quanto tempo (espresso in secondi) i diversi tool riescono a fornire informazioni sulla diagnosticabilità del sistema dato, ed inoltre possiamo fare delle considerazioni sulla cardinalità dello spazio di stato. Nel Capitolo 6 vedremo alcuni esempi per evidenziare i vantaggi di utilizzo di un toolbox rispetto ad un altro.

Per prima cosa però, è stato necesserio testare la funzionalità della piattaforma in riferimento ai toolbox realizzati da Pocci e Perria. Per questo motivo abbiamo deciso di seguire due diverse strade, la prima è mostrata in Figura 5.1 e la seconda nella Figura 5.2. In entrambe le figure, si è usata la seguente notazione:

- i rettangoli indicano i file;
- i rombi indicano che si sta utilizzando un toolbox;
- gli archi indicano i *plugin* e gli *adapter* utilizzati per le conversioni.

Nella prima possiamo vedere come partendo da una Rete di Petri disegnata utilizzando il tool *Pipe3* sia possibile passare tramite l'*adapterA15_Pipe3toPipe2* ad una rete nel formato consentito da *Pipe2* (anche se sia *Pipe3* che *Pipe2* generano file con estensione .xml). Dalla rete in *Pipe2* grazie al toolbox *Tina* viene generato il grafo di raggiungibilità della Rete di Petri (tramite il comando: $tina.exe -R -PNML\ file\_pipe2.xml\ file\_reach\_graph.txt$) contenuto nel file con estensione .txt. Il grafo di raggiungibilità non è altro che l'automa corrispondente alla rete di Petri iniziale, quindi utilizziamo il *pluginP7* per ottenere l'automa nel formato compatibile con *Desuma*, ovvero avremo un file con estensione .fsm. Una volta ottenuto tale file facciamo uso dell'eseguibile *diag_UR* per visualizzare il diagnosticatore e di *dcycle* per avere le informazioni riguardanti la presenza o meno di cicli indeterminati all'interno del sistema considerato e di conseguenza stabilire se la rete considerata è diagnosticabile oppure no. I risultati ottenuti con *Desuma* li confrontiamo con quelli uscenti dai toolbox *PN_DIAG* e *PN_DIAG_2*. Per poter utilizzare questi ultimi due, sempre a partire dalla Rete di Petri disegnata con *Pipe3*, si ha la necessità di utilizzare prima il *pluginP1* (che fornisce un file di estensione .pnml), poi l'*adapterA1* (che dal formato di estensione .pnml passa al formato Matlab_DISC di estensione .m) e infine si usa l'*adapterA9* (che dal formato Matlab_DISC passa al formato PN_DIAG_Matlab) che restituisce il file di estensione .m che caratterizza la RdP, e quest'ultimo file generato corrisponde al file da mandare in input ai toolbox, in modo da avere tutte le informazioni riguardanti la diagnosticabilità.

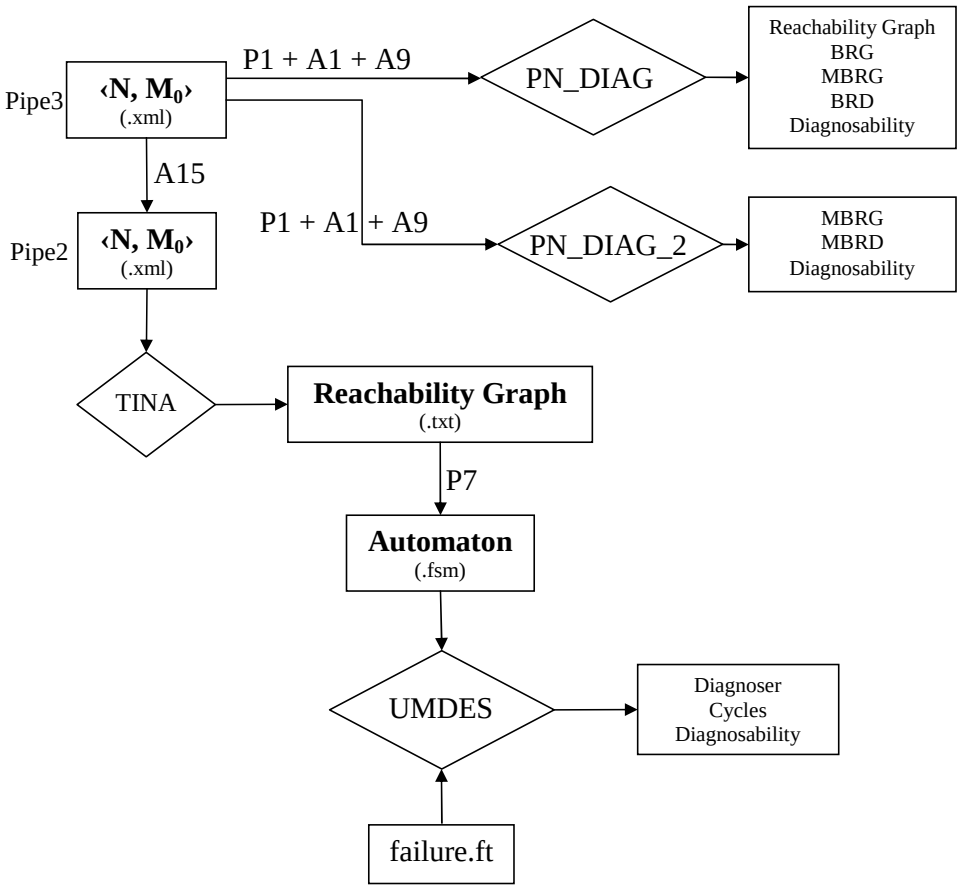

Figura 5.1: Test dei *plugin* e degli *adapter* a partire da una Rete di Petri disegnata con Pipe3

Nella seconda invece, si ha il procedimento inverso, ovvero, a partire dal file di estensione .m possiamo immediatamente mandarlo in ingresso ai toolbox di Pocci e Perria, e subito si ottengono le informazioni sulla diagnosticabilità, ma per poter fare il confronto con i risultati forniti da *Desuma* si ha la necessità di usare il *pluginP9* (che permette di passare dal formato PN_DIAG_Matlab al formato Matlab_DISC), poi usiamo il *pluginP2* (che permette di passare dal formato Matlab_DISC al formato .pnml), poi l'*adapterA2* (che permette di passare dal formato .pnml al formato .xml), subito dopo l'*adapterA15_Pipe3toPipe2* (che permette di passare dal formato .xml di *Pipe3* all' .xml di *Pipe2*), poi grazie a *Tina* otteniamo il grafo di raggiungibilità della rete e infine il *pluginP7* genera l'automa di cui si vuole analizzare la diagnosticabilità, quest'ultima fase viene effettuata attraverso i file eseguibili *diag_UR* e *dcycle* che generano ri-

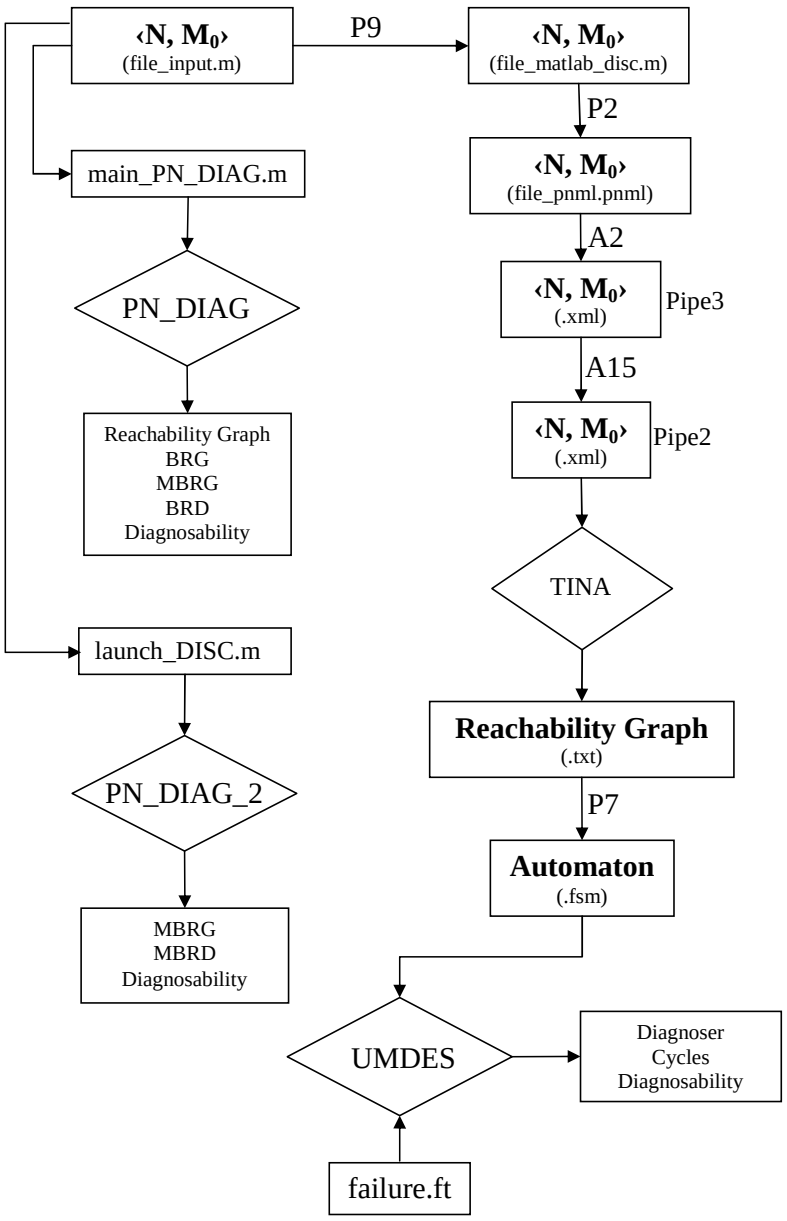

Figura 5.2: Test dei *plugin* e degli *adapter* a partire da una Rete di Petri caratterizzata da $(Pre, Post, M_0, F, L, E)$

spettivamente il diagnosticatore e le informazioni sulla presenza o meno di cicli indeterminati, e quindi definire se il sistema è diagnosticabile oppure no.

Naturalmente i risultati dei toolbox *PN_DIAG*, *PN_DIAG_2* e *Desuma*, per quanto riguarda la diagnosticabilità devono essere concordi, in caso contrario vuol dire che c'è qualche *bug* da correggere nei toolbox.

Poichè nella *Piattaforma Software* è inserito anche il toolbox *Hypens* sono stati testati anche i *plugin* e gli *adapter* che lo riguardano.

5.2 Problemi durante l'uso della Piattaforma Software

In questa sezione verranno illustrati i problemi riscontrati durante la fase di testing della *Piattaforma Software* e dei tool con cui essa collabora. Accenniamo subito un problema riscontrato seguendo entrambe le strade citate sopra. Mandando in esecuzione gli script, i risultati trovati vengono scritti nell'output console (in basso nella schermata della piattaforma). Se quello che lo script richiede di fare è un'operazione piccola non c'è nessun tipo di problema, tutto viene correttamente visualizzato, ma se il lavoro da fare è un po' più consistente, nell'output console viene visualizzata soltanto una parte dei risultati, questo è dovuto a un problema di buffer, ovvero lo spazio di memoria a disposizione della piattaforma non è sufficiente per registrare tutti i dati risultanti. Questo problema si può aggirare aggiungendo nello script un piccolo codice Matlab che permetta di salvare i risultati che occorrono in un file di testo, così da poterli esaminare in un secondo momento. Vediamo gli altri problemi incontrati più in dettaglio.

5.2.1 Problemi nel tool Pipe3

Un primo problema è stato riscontrato nell'utilizzo del toolbox *Pipe3* nel momento in cui si doveva disegnare la rete mostrata in Figura 5.3, che ha come insieme delle transizioni osservabili $T_o = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$, insieme delle transizioni non osservabili $T_u = \{\varepsilon_8, \varepsilon_9, \varepsilon_1 0, \varepsilon_{11}, \varepsilon_{12}\}$ e insieme delle transizioni di guasto $T_f = T_f^1 \cup T_f^2$, dove $T_f^1 = \{\varepsilon_{11}\}$ e $T_f^2 = \{\varepsilon_{12}\}$, per quanto riguarda le etichette associate alle transizioni $\mathscr{L}(t_1) = a$, $\mathscr{L}(t_2) = \mathscr{L}(t_3) = b$, $\mathscr{L}(t_4) = \mathscr{L}(t_5) = c$, $\mathscr{L}(t_6) = \mathscr{L}(t_7) = d$. Il problema è dovuto al fatto che si hanno transizioni con la stessa etichetta e *Pipe3* non supporta questa possibilità. Per ovviare a questo problema, è stato fatto il seguente passaggio, si è disegnata la rete in Figura 5.3 tramite il toolbox *Pipe2* che invece permetteva di avere più transizioni con uguale etichetta, salvare tale rete e poi riaprire il file creato facendo uso del toolbox

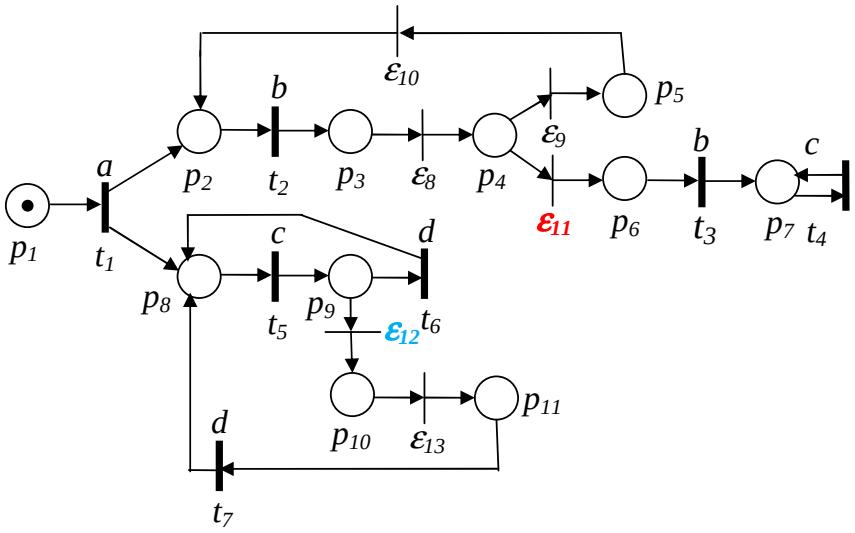

Figura 5.3: Esempio di Rete di Petri con transizioni aventi la stessa etichetta e con più di nove transizioni

Pipe3 e salvarla in tale toolbox. Inoltre, facendo uso del toolbox *Pipe3* si è notato che quando vengono inseriti i posti e le transizioni, la loro numerazione parte da 0, ovvero si ha il posto p_0 e la transizione t_0 e così via. Questa numerazione va in conflitto con i toolbox *PN_DIAG* e *PN_DIAG_2*, perchè per come sono stati realizzati hanno la necessità di avere i posti e le transizioni che partano dal numero 1, in caso contrario forniscono risultati sulla diagnosticabilità scorretti. Per ovviare a questo problema, si è utilizzato un piccolo artificio, al momento di creare la rete si inseriscono un numero di posti pari a quello della rete da disegnare più uno, stessa cosa vale per le transizioni, i collegamenti anzichè partire dal posto p_0 (cioè il primo inserito) li facciamo iniziare dal posto p_1, e il posto p_0 viene cancellato, stessa cosa si effettua per la transizione t_0.

5.2.2 Problemi nel tool Desuma

Poichè uno degli scopi di questa tesi è mettere a confronto i risultati forniti dai software realizzati da Pocci e Perria, ed inoltre confrontare i risultati forniti dai precedenti con quelli generati da *Desuma*, è stato necessario verificare che anche quest'ultimo funzionasse correttamente, in particolare, si sono testati i file eseguibili *diag_UR* e *dcycle*, facenti parte della libreria UMDES. Questi programmi sono stati testati in numerose RdP, tra cui quella in Figura 5.3 e il risul-

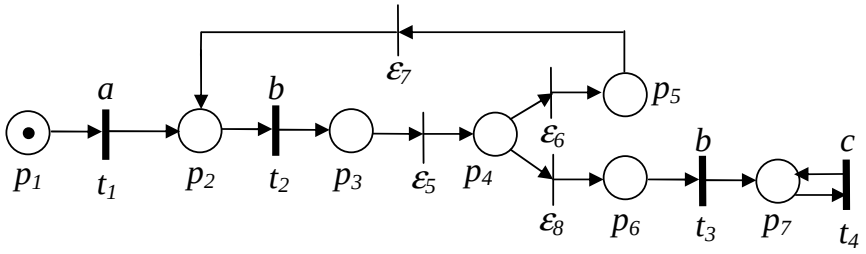

Figura 5.4: Esempio di Rete di Petri che provoca un baco in *Desuma*

PN_DIAG										
x	$\|R\|$	$t_R[s]$	$\|BRG\|$	$t_{BRG}[s]$	$\|MBRG\|$	$t_{MBRG}[s]$	$\|BRD\|$	$t_{BRD}[s]$	$t_{diag}[s]$	$Diag?$
1	7	0.0398	4	0.1123	6	0.0435	5	0.1711	0.3697	si

Tabella 5.1: Risultati della simulazione effettuata con il toolbox *PN_DIAG* prendendo in considerazione la rete in Figura 5.4

tato fornito è stato coerente con quello fornito dai toolbox di Pocci e Perria, per quando riguarda la diagnosticabilità. Mentre i problemi sono sorti nella rete in Figura 5.4.

Facciamo degli esempi per spiegare meglio il problema.

Esempio 5.2.1. *Consideriamo la Rete di Petri in Figura 5.4, dove l'insieme delle transizioni osservabili è* $T_o = \{t_1, t_2, t_3, t_4\}$*, l'insieme delle transizioni non osservabili è* $T_u = \{\varepsilon_5, \varepsilon_6, \varepsilon_7, \varepsilon_8\}$*, l'insieme delle transizioni di guasto è* $T_f = T_f^1 \cup T_f^2$*, dove* $T_f^1 = \{\varepsilon_5\}$ *e* $T_f^2 = \{\varepsilon_8\}$*, le etichette associate alla transizioni sono:* $\mathscr{L}(t_1) = a$*,* $\mathscr{L}(t_2) = \mathscr{L}(t_3) = b$ *e* $\mathscr{L}(t_4) = c$*.*

I risultati forniti dal toolbox PN_DIAG *sono riportati in Tabella 5.1 (strutturata come la Tabella 4.1), come si può vedere il sistema risulta diagnosticabile.*

I risultati forniti dal toolbox PN_DIAG_2 *sono riportati in Tabella 5.2 (strutturata come la Tabella 4.2), come si può vedere il sistema risulta diagnosticabile, in accordo con quanto detto dal toolbox* PN_DIAG*.*

Mentre i risultati forniti da Desuma *utilizzato all'interno della* Piattaforma Software*, tramite il comando* diag_UR *genera un file contenente il diagnosticatore dell'automa, e con il comando* dcycle *genera un file che indica se sono presenti nel diagnosticatore cicli indeterminati. I risultati ottenuti sono riportati nella*

PN_DIAG_2								
x	$\|MBRG\|$	$t_{MBRG}[s]$	$\|MBRD\|$	$t_{MBRD}[s]$	*classe guasto*	*# c. incerti*	$t_{diag}[s]$	*Diag?*
1	6	0.1518	5	0.0697	una due	0 1	0.3506	si

Tabella 5.2: Risultati della simulazione effettuata con il toolbox *PN_DIAG_2* prendendo in considerazione la rete in Figura 5.4

Desuma							
# nodi	$t_R[s]$	*# stati diag*	$t_{diag}[s]$	*# cicli indet*	$t_{cicli}[s]$	$t_{UMDES}[s]$	*Diag?*
7	0.0424	5	0.0582	1	0.0393	0.0397	no

Tabella 5.3: Risultati della simulazione effettuata con il toolbox *Desuma* prendendo in considerazione la rete in Figura 5.4

Tabella 5.3, strutturata nel seguente modo:

- *la colonna* 1 *indica il numero di nodi dell'automa a stati finiti (*$\#\ nodi$*);*
- *la colonna* 2 *indica il tempo di calcolo espresso in secondi per generare l'automa (*$t_R[s]$*);*
- *la colonna* 3 *indica il numero di stati del diagnosticatore (*$\#\ stati\ diag$*);*
- *la colonna* 4 *indica il tempo di calcolo del diagnosticatore espresso in secondi (*$t_{diag}[s]$*);*
- *la colonna* 5 *indica il numero di cicli indeterminati trovati tramite* dcycle *(*$\#\ cicli\ indet$*);*
- *la colonna* 6 *indica il tempo di calcolo espresso in secondi per la ricerca di cicli indeterminati (*$t_{cicli}[s]$*);*
- *la colonna* 7 *indica il tempo totale espresso in secondi per la verifica della diagnosticabilità (*$t_{UMDES}[s]$*);*
- *la colonna* 8 *indica se il sistema è diagnosticabile oppure no.*

Come si può vedere dcycle *genera un file che contiene un ciclo indeterminato, ma questo non può essere vero. Abbiamo verificato allora se il diagnosticatore prodotto da* diag_UR *fosse corretto. Riportiamo in Figura 5.5 l'automa di cui*

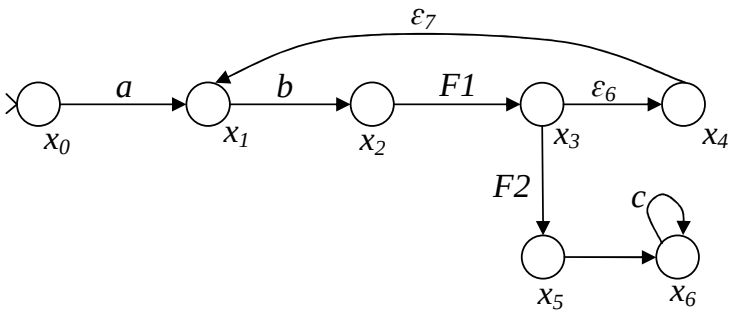

Figura 5.5: Automa corrispondente alla Rete di Petri in Figura 5.4

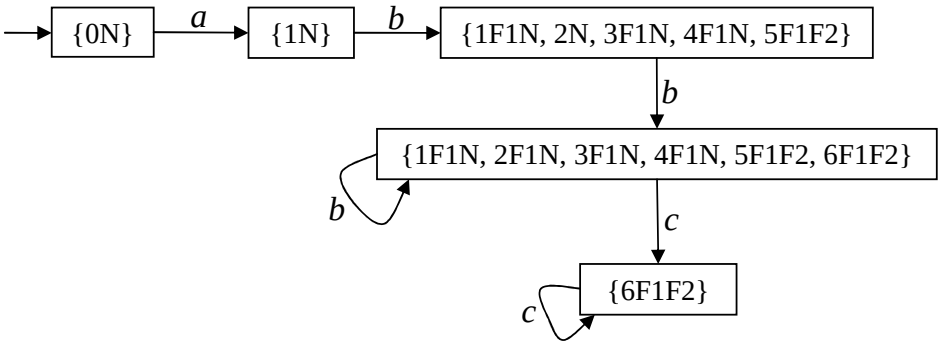

Figura 5.6: Diagnosticatore corrispondente all'automa in Figura 5.5

vogliamo calcolare il diagnosticatore, in Figura 5.6 il diagnostiscatore e in Figura 5.7 il file generato da dcycle*. Nelle Figure 5.5 e 5.6 si è indicato con* $F1$ *la transizione di guasto* $\varepsilon_5 \in T_f^1$ *e con* $F2$ *la transizione di guasto* $\varepsilon_8 \in T_f^2$*.*

Analizzando i risultati prodotti è evidente che il bug *è presente in* dcycle*, perchè le informazioni fornite nel file generato da esso sono errate (Figura 5.7), vede un ciclo indeterminato che non c'è, infatti, sia l'automa che il diagnosticatore sono corretti. Il problema potrebbe essere generato dal loop presente nello stato* x_6*.* ■

Sempre considerando la rete in Figura 5.4 lo stesso errore si presente nei seguenti casi:

```
dcycle: Indeterminate cycle(s):
Format: {D_state} {event} -> {next D_state} {event} ->...

4 b -> 4
Uncertain: F2
Diagnoser State 4 contains the following FSA states:
S2
S6
```

Figura 5.7: File generato da *dcycle* per il diagnosticatore in Figura 5.6

1. $T_f = T_f^1 \cup T_f^2 \cup T_f^3$, con $T_f^1 = \{\varepsilon_5\}$, $T_f^2 = \{\varepsilon_7\}$ e $T_f^3 = \{\varepsilon_8\}$;
2. $T_f = T_f^1 \cup T_f^2$, con $T_f^1 = \{\varepsilon_7\}$ e $T_f^2 = \{\varepsilon_8\}$.

Mentre la soluzione fornita da *Desuma* è corretta in questi altri casi:

1. $T_f = \{\varepsilon_5\}$;
2. $T_f = \{\varepsilon_7\}$;
3. $T_f = \{\varepsilon_8\}$;
4. $T_f = \{\varepsilon_5, \varepsilon_7\}$;
5. $T_f = \{\varepsilon_5, \varepsilon_8\}$;
6. $T_f = \{\varepsilon_7, \varepsilon_8\}$;
7. $T_f = \{\varepsilon_5, \varepsilon_7, \varepsilon_8\}$.

Da questo si può dedurre che il problema è dovuto alla presenza della transizione di guasto ε_8 nel caso in cui l'insieme delle transizioni di guasto sia costituito da più classi di guasto, mentre non si crea nessun tipo di errore nel caso in cui la transizione di guasto ε_8 appartenga ad un'unica classe di guasto anche in presenza di altre transizioni di guasto (ad esempio insieme alle transizioni di guasto ε_5 e ε_7).

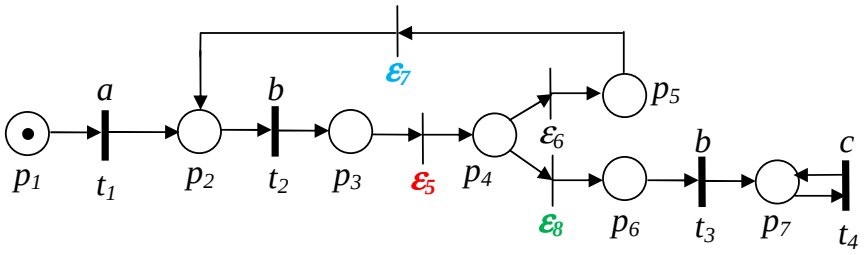

Figura 5.8: Esempio di Rete di Petri con tre classi di guasto

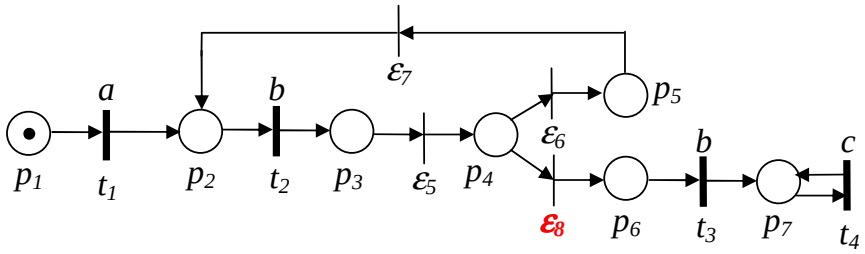

Figura 5.9: Esempio di rete di Petri con una classe di guasto

Una volta constatato questo errore, abbiamo contattato il professore Lafortune, che aveva coordinato insieme a Teneketzis, il gruppo *Discrete Event System* dell'università del Michigan per lo sviluppo del software UMDES, in modo che risolvano questo *bug*.

5.2.3 Test numero uno sui plugin

In questa sezione elenchiamo i problemi riscontrati nell'utilizzo dei *plugin* realizzati. Per testare i *plugin* abbiamo utilizzato le reti di Petri in Figura 5.3, 5.8 e 5.9, per le ultime due reti l'insieme delle transizioni osservabili è $T_o = \{t_1, t_2, t_3, t_4\}$, l'insieme delle transizioni non osservabili è $T_u = \{\varepsilon_5, \varepsilon_6, \varepsilon_7, \varepsilon_8\}$, le etichette associate alle transizioni sono $\mathscr{L}(t_1) = a$, $\mathscr{L}(t_2) = \mathscr{L}(t_3) = b$ e $\mathscr{L}(t_4) = c$, invece, per quanto riguarda l'insieme delle transizioni di guasto nella rete in Figura 5.8: $T_f = T_f^1 \cup T_f^2 \cup T_f^3$, dove $T_f^1 = \{\varepsilon_5\}$, $T_f^2 = \{\varepsilon_7\}$ e $T_f^3 = \{\varepsilon_8\}$ (quindi tre diverse classi di guasto, ognuna contenente una transizione di guasto) e per la rete in Figura 5.9 $T_f = \{\varepsilon_8\}$ (quindi una sola classe di guasto contenente una sola transizione di guasto).

Il *pluginP1.exe* permette di convertire una rete di Petri disegnata tramite *Pipe3* nel formato Pnml_DISC. Tale *plugin* funziona correttamente in tutte le reti provate.

Il *pluginP2.exe* permette di convertire il formato Matlab_DISC nell'Pnml_DISC. Anche questo è stato provato nelle reti di Petri in Figura 5.3, 5.8 e 5.9. La soluzione fornita per tutte le reti è scorretta in quanto la numerazione dei posti parte da p_0 e per quanto riguarda le transizioni non vengono riconosciute le etichette (ovvero a, b, c) ed inoltre vengono numerate a partire da t_0.

Il *pluginP3.exe* permette di convertire un qualsiasi grafo in formato .fsm (di *Desuma*) nel formato Pnml_DISC. È stato testato nelle reti di Petri in Figura 5.3, 5.8 e 5.9. Nel caso in cui il grafo in formato .fsm deriva dal grafo di raggiungibilità generato con *Tina* il *pluginP3* non funziona correttamente, in quanto *Tina* nomina gli stati a partire da 0 e di conseguenza il grafo in formato .fsm (generato dal *pluginP7*) parte anch'esso da 0 provocando un mal funzionamento nel *pluginP3*. Si è provato allora a modificare il grafo in formato .fsm facendo partire la numerazione dei posti da 1, e in questo modo il file in formato .pnml fornito dal *pluginP3* è corretto, questo è vero però solo per la rete che ha una transizione di guasto in ogni classe di guasto. Spieghiamo meglio, il *pluginP3* è stato testato nella rete in Figura 4.1, dove l'insieme delle transizioni di guasto è $T_f = \{\varepsilon_5, \varepsilon_7\}$, ovvero contiene due transizioni di guasto in un'unica classe di guasto ed in questo caso il risultato fornito è errato. Testando il *plugin* nelle rete in Figura 5.3 il risultato fornito è comunque errato, nonostante ci siano due classi di guasto ognuna contenente una sola transizione di guasto, l'errore è dovuto alla presenza di un numero di transizioni maggiori di nove. Mentre nelle reti in Figura 5.8 e 5.9 ha fornito risultati corretti, infatti sia nella prima che nella seconda rete ogni classe di guasto contiene una sola transizione di guasto e il numero delle transizioni è inferiore a nove.

Il *pluginP7.exe* permette di convertire un qualsiasi grafo in formato .txt generato da *Tina* in un altro grafo nel formato .fsm, ovvero quello di *Desuma*. Tale *plugin* non funziona correttamente se nella rete di Petri ci sono più di nove transizioni, quindi nel caso della rete in Figura 5.3 il grafo di raggiungibilità fornito dal *pluginP7* è del tutto sbagliato, mentre nelle reti in Figura 5.8 e 5.9 viene generato un grafo perfettamente concorde con quello generato da *Tina*, però come già accennato in precedenza, *Tina* nomina gli stati a partire da 0 e così fa anche il *pluginP7*. A fronte di quanto detto, sarebbe opportuno modificarlo in modo che gli stati del grafo, quindi dell'automa di cui si vorrà studiare la diagnosticabilità vengano numerati a partire da 1, così che i risultati prodotti possano venire tranquillamente usati anche dal *pluginP3*.

Il *pluginP8.exe* permette di convertire il file Matlab che viene mandato in ingresso al toolbox *Hypens* realizzato da Sessego [32] nel formato Matlab_DISC. Il file.m restituito non è quello desiderato, le matrici Pre e $Post$ generate non sono quelle attese.

Il *pluginP9.exe* permette di convertire il file Matlab che viene mandato in ingresso ai toolbox di Pocci e Perria nel formato Matlab_DISC. Non è stato possibile testarne l'efficacia perchè si blocca ancora prima di fare qualsiasi cosa.

5.2.4 Test numero uno sugli adapters

In questa sezione elenchiamo i problemi riscontrati nell'utilizzo degli *adapters* realizzati. Per testare gli *adapters* abbiamo utilizzato le stesse reti di Petri usate per il test dei *plugin*, ovvero quelle in Figura 5.3, 5.8 e 5.9. Sono state scelte queste reti perchè ognuna ha una caratteristica diversa che permette di testare l'effettivo funzionamento.

L'*adapterA1.exe* permette di convertire il formato Pnml_DISC nel formato Matlab_DISC (ossia fa l'operazione inversa del *pluginP2*). Anche questo *adapter* è stato provato nelle reti in Figura 5.3, 5.8 e 5.9. Il risultato fornito per la rete in Figura 5.3 è scorretto, in particolare sono errate le matrici Pre, $Post$ e il vettore di celle F, gli errori sono dovuti alla presenza di un numero di transizioni maggiore di nove e al fatto che sono presenti più classi di guasto. Nella rete in Figura 5.8 sono corrette le matrici Pre e $Post$ ma il vettore di celle F non tiene conto della presenza di più classi di guasto, mentre nella rete in Figura 5.9 il risultato fornito è corretto, essendoci una sola classe di guasto e meno di nove transizioni.

L'*adapterA2.exe* permette di convertire il formato Pnml_DISC nel formato .xml, ovvero fornisce la Rete di Petri visualizzabile mendiante il tool *Pipe3*. Ha funzionato perfettamente in tutte le reti di Petri provate.

L'*adapterA3.exe* permette di convertire il formato Pnml_DISC nel formato .fsm (di *Desuma*), ovvero fornisce il grafo di raggiungibilità della rete corrispondente e di conseguenza l'automa. Non è stato possibile testare tale *adapter* dentro la piattaforma, poichè mandando in esecuzione la conversione, anziché partire l'*adapterA3*, vengono eseguiti l'*adapterA2*, l'*adapterA15_Pipe3toPipe2*, il tool *Tina* e il *pluginP7*, il file .fsm generato è congruente con il grafo generato da *Tina* con il problema segnalato in precedenza, ovvero sarebbe necessario apportare una modifica al *pluginP7* in modo che gli stati dell'automa vengano numerati a partire da uno. Si è proceduto allora a testare l'*adapterA3* fuori dalla piattaforma

(cioè dal prompt dei comandi) ma il risultato fornito per tutte le reti provate è errato, infatti nel grafo non compaiono i nomi delgi eventi e il numero degli stati dell'automa è inferiore a quello che si dovrebbe ottenere.

L'*adapterA8.exe* permette di convertire il formato Matlab_DISC nel formato Matlab_Hypens, così da avere il file.m da mandare in ingresso al toolbox *Hypens*. Il file.m generato risulta essere corretto, per quanto riguarda alcuni campi, mentre per altri, in particolare per la matrice D e il vettore s ci sono dei problemi, nonostante abbiano entrambi dimensione corretta. Per poter utilizzare il tool *Hypens*, è necessario intervenire nel file generato dall'*adapterA8* prima di lanciare l'esecuzione del tool. In particolare, per la matrice D è necessario indicare i tempi (espressi in secondi) associati a ogni transizione, in modo da evitare che tentino di scattare prima transizioni non abilitate, bloccando così la simulazione. Per il vettore s, che associa a ogni transizione la politica di servente, è necessario quindi decidere quali transizioni possono scattare un numero infinito di volte, quali solo una volta e quali k-volte. Gli sviluppatori stanno pensando ad un modo per automatizzare l'operazione.

L'*adapterA9.exe* permette di convertire il formato Matlab_DISC nel formato PN_Diag_Matlab, ovvero il file da mandare in ingresso ai toolbox di Pocci e Perria. Tale *adapter* ha restituito le matrici Pre e $Post$ errate e non riesce a riconoscere la presenza di più classi di guasto. Il problema però, è dovuto principalmente al fatto che l'*adapterA9* viene usato sempre in seguito all'*adapterA1*, trascinandosi dietro gli errori prodotti da esso. Infatti ha funzionato correttamente solo nella rete in Figura 5.9 ovvero dove è presente una sola classe di guasto e il numero di transizioni è inferiore a nove. Nella rete in Figura 5.3 ha prodotto un risultato errato, sia in termini di matrici Pre e $Post$ sia per quanto riguarda il vettore di celle F, anzichè avere $F = \{[11];[12]\}$, ha fornito il vettore $F = \{[11,12]\}$, quindi inserendo le transizioni di guasto in un unica classe. Per la rete in Figura 5.8 il file.m generato è corretto per le matrici Pre e $Post$, mentre è errato per il vettore di celle F, avendo tre classi di guasto.

L'*adapterA15_Pipe3toPipe2.exe* permette di convertire la rete di Petri disegnata con *Pipe3* in una RdP nel formato di *Pipe2*. Tale *adapter* è stato creato per poter utilizzare *Tina* all'interno della piattaforma, in quanto al momento accetta soltanto RdP disegnate con *Pipe2*. In tutti i casi provati (Figure 5.3, 5.8, 5.9) questo *adapter* ha funzionato sempre correttamente.

5.2.5 Test numero due sui plugin

In questa sezione verranno elencati i problemi incontrati nell'utilizzo dei *plugin* in seguito alla revisione da parte dell'ingegnere Contini di Akhela srl. Verranno riportati i problemi solo dei *plugin* sottoposti a revisione.

Il *pluginP2.exe* che converte il formato Matlab_DISC nel formato Pnml_DISC, restituisce un file .pnml ma non vengono riconosciuti i nomi delle transizioni non osservabili, siano esse di guasto oppure no. Inoltre nell'elenco degli archi viene riportato il posto p_0 e la transizione t_0, cosa del tutto errata, in quanto quel posto e quella transizione non sono neppure riportati nell'elenco dei posti e delle transizioni, rispettivamente. Questi problemi sono stati riscontrati per tutte le reti prese in considerazione, ovvero quelle in Figura 5.3, 5.8 e 5.9.

Il *pluginP3.exe* che converte un qualsiasi grafo in formato .fsm (di *Desuma*) nel formato Pnml_DISC non funziona correttamente in quanto restituisce si un file in formato .pnml, ma in esso non vengono riconosciuti i nomi delle transizioni, siano essere osservabili oppure no.

Il *pluginP7.exe* che converte un qualsiasi grafo in formato .txt generato da *Tina* in un altro grafo nel formato .fsm, ovvero quello di *Desuma*, restituisce un file .fsm errato. Spieghiamo meglio in che cosa consistono gli errori. L'automa generato è giusto per quanto riguarda la struttura, ovvero i nodi sono collegati tramite gli archi nel modo corretto, ma l'errore sta nell'etichetta associata agli archi, facciamo un esempio per chiarire la situazione, osservando la rete in Figura 5.9, il grafo di raggiungibilità generato da *Tina* si può osservare in Figura 5.10, dove lo stato 0 corrisponde ad una marcatura nel posto p_1, lo stato 1 ad una marcatura nel posto p_2, lo stato 2 ad una marcatura nel posto p_3, lo stato 3 ad una marcatura nel posto p_4, lo stato 4 ad una marcatura nel posto p_5, lo stato 5 ad una marcatura nel posto p_6 e infine lo stato 6 ad una marcatura nel posto p_7. Inoltre conosciamo l'etichetta corrispondente a ciascuna transizione, ovvero sappiamo che a t_1 corrisponde l'etichetta a, a t_2 e t_3 l'etichetta b, a t_4 l'etichetta c e tutte le altre sono non osservabili, si riconoscono dalla E prima del nome, inoltre la F indica che la transizione è di guasto. Dal grafo di raggiungibilità utilizzando il *pluginP7* si ottiene l'automa in Figura 5.11, come si nota è impossibile passare dallo stato s_1 allo stato s_2 con l'arco etichettato F_8, e così via per gli altri errori. Questo problema è stato riscontrato in tutte le reti provate, indipendente dal numero di transizioni e dal numero di classi di guasto. Per scrupolo è stato testato anche nella rete in Figura 5.12, dove tutte le transizioni sono osservabili, quindi $T_o = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}, t_{13}\}$ e le etichette associate alle transizioni sono $\mathscr{L}(t_1) = \mathscr{L}(t_{10}) = \mathscr{L}(t_{12}) = a$, $\mathscr{L}(t_2) = \mathscr{L}(t_3) = \mathscr{L}(t_{13}) = b$, $\mathscr{L}(t_4) = \mathscr{L}(t_5) = \mathscr{L}(t_8) = \mathscr{L}(t_9) = c$ e $\mathscr{L}(t_6) = \mathscr{L}(t_7) = \mathscr{L}(t_{11}) = d$, ma il fi-

```
INPUT NET ----------------

parsed net noname

7 places, 8 transitions

net noname
tr t1 : a p1 -> p2
tr t2 : b p2 -> p3
tr t3 : b p6 -> p7
tr t4 : c p7 -> p7
tr t5 : E5 p3 -> p4
tr t6 : E6 p4 -> p5
tr t7 : E7 p5 -> p2
tr t8 : F8 p4 -> p6
pl p1 : P1 (1)
pl p2 : P2
pl p3 : P3
pl p4 : P4
pl p5 : P5
pl p6 : P6
pl p7 : P7
```

```
REACHABILITY ANALYSIS ------------

bounded

7 marking(s), 8 transition(s)

MARKINGS:

0 : p1
1 : p2
2 : p3
3 : p4
4 : p5
5 : p6
6 : p7

REACHABILITY GRAPH:

0 -> t1/1
1 -> t2/2
2 -> t5/3
3 -> t6/4, t8/5
4 -> t7/1
5 -> t3/6
6 -> t4/6
```

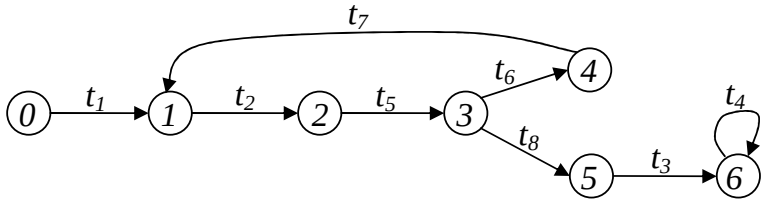

Figura 5.10: Grafo di raggiungibilità generato da *Tina* per la rete in Figura 5.9

le.fsm generato è risultato non concorde con quello generato da *Tina*, rivelandosi così errato.

Il *pluginP9.exe* che converte il file nel formato PN_Diag_Matlab nel Matlab_DISC ha restituito per la reti in Figura 5.3, 5.8 e 5.9 un file.m che rappresenta il formato Matlab_DISC. E' stato possibile testare tale *plugin* solo attraverso il prompt dei comandi, all'interno della *Piattaforma Software* al momento della conversione compare una finestra di errore per informare che poichè il formato in ingresso non è stato riconosciuto, la conversione è stata bloccata. Il formato Matlab_DISC possiamo dividerlo in sei parti:

1. matrici della rete;

   (i) la matrice Pre di m righe, tante quanti sono i posti della rete e n

```
7

S0      0       1
a       S1      c       o

S1      0       1
F8      S2      c       uo

S2      0       1
E5      S3      c       uo

S3      0       2
E6      S4      c       uo
F8      S5      c       uo

S4      0       1
E7      S1      c       uo

S5      0       1
E7      S6      c       uo

S6      0       1
E6      S6      c       uo
```

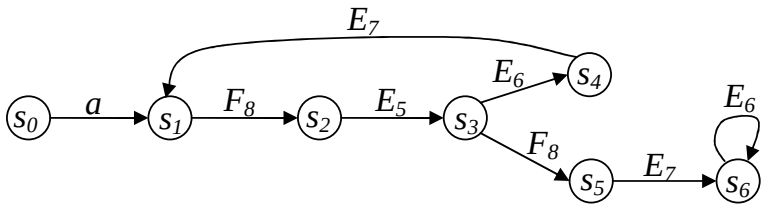

Figura 5.11: Automa errato riferito alla Figura 5.9 ottenuto utilizzando il *pluginP7*

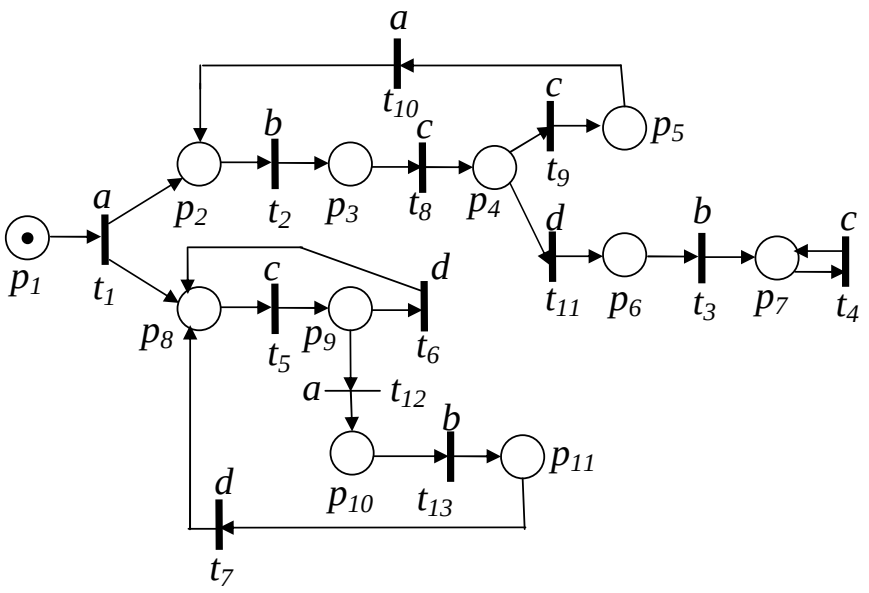

Figura 5.12: Esempio di rete di Petri contenente solo transizioni osservabili

colonne, tante quante sono le transizioni. In seguito all'esecuzione del *pluginP9* per le reti di Figura 5.3, 5.8 e 5.9 la matrice Pre è corretta;

(ii) la matrice $Post$ di m righe, tante quanti sono i posti della rete e n colonne, tante quante sono le transizioni. In seguito all'esecuzione del *pluginP9* per le reti di Figura 5.3, 5.8 e 5.9 la matrice $Post$ è corretta;

(iii) il vettore M_0 di m righe, tante quanti sono i posti della rete e 1 colonna, indica la marcatura iniziale della RdP in considerazione. In seguito all'esecuzione del *pluginP9* tale vettore è stato convertito in modo corretto per tutte le RdP provate, comprese le reti in Figura 5.3, 5.8 e 5.9;

2. informazioni per il toolbox *Hypens*, che al momento sono settate con valori di default; per lo scopo di questa tesi possono essere lasciati inalterati perchè non influenzano il funzionamento dei toolbox *PN_DIAG* e *PN_DIAG_2*;

3. informazioni per i toolbox *PN_DIAG* e *PN_DIAG_2*

(i) il vettore F di 1 riga e n colonne, tante quante sono le transizioni della rete, è il vettore che indica quali sono le transizioni di guasto, associa il valore -1 alla transizioni osservabili e non osservabili (ma non di guasto) e il valore 1 a quelle di guasto. Se le classi di guasto sono più di una, associa alla transizione il numero della classe di guasto corrispondente. In seguito all'esecuzione del *pluginP9* il vettore non è quello atteso, in quanto in corrispondenza delle transizioni di guasto non si ha il numero indicante la classe di guasto e tra l'altro non riconosce la presenza di un numero di classi di guasto maggiore di uno. In particolare, per la rete in Figura 5.3 indica come transizioni di guasto ε_{12} e ε_{13}, riconoscendole come appartenenti ad un'unica classe, mentre sono ε_{11} e ε_{12} appartenenti rispettivamente alla classe di guasto 1 e alla classe di guasto 2. Per la rete in Figura 5.8, indica come transizioni di guasto ε_6 e ε_8, riconoscendole come appartenenti ad un'unica classe, mentre sono ε_5 e ε_7 e ε_8 appartenenti rispettivamente alla classe di guasto 1, 2 e 3. Per la rete in Figura 5.9 non indica la presenza di nessuna transizione di guasto, tutti i suoi elementi sono pari a -1, invece l'elemento nella colonna 8 sarebbe dovuto essere pari a 1, essendo ε_8 la transizione di guasto;

(ii) il vettore L di 1 riga e n colonne, tante quante sono le transizioni. In ogni colonna vi è l'etichetta associata alla transizione seguita da tante & quanto il numero delle etichette diverse tra loro più il numero di transizioni non osservabili (comprese quelle di guasto). In seguito all'esecuzione del *pluginP9* anche questo vettore si è rivelato errato per tutte le reti testate. Per la rete di Figura 5.3 non ha riconosciuto che $\mathscr{L}(t_3) = b$ e $\mathscr{L}(t_5) = c$. Per la rete in Figura 5.8 che $\mathscr{L}(t_3) = b$, stesso risultato si è ottenuto per la rete in Figura 5.9;

4. informazioni generiche sui posti e sulle transizioni

(i) il vettore P_{index} di m colonne, tante quanti sono i posti della rete e 1 riga e il vettore T_{index} di n colonne, tante quante sono le transizioni della rete e 1 riga. In seguito all'esecuzione del *pluginP9* per tutte le reti provate i vettori restituiti sono vuoti, potrebbe rappresentare un problema, in quanto P_{index} fornisce in quale ordine sono considerati i posti e di conseguenza permette di sapere se la matrice Pre è corretta e T_{index} fornisce in quale ordine sono considerate le transizioni e di conseguenza permette di sapere se la matrice $Post$ è corretta;

(ii) il vettore P_t di m colonne, tante quanti sono i posti della rete e 1 riga e il vettore T_t di n colonne, tante quante sono le transizioni della rete e 1 riga. In seguito all'esecuzione del *pluginP9* per tutte

le reti provate i vettori restituiti sono vuoti, mentre avrebbero dovuto indicarci che tutti i posti e le transizioni sono discreti (D), in quanto stiamo considerando RdP discrete;

5. informazioni più specifiche sui posti e sulle transizioni, riportiamo solo i vettori di interesse per la tesi

   (i) i vettori P_{obs} e P_{contr} entrambi di m colonne, tante quanti sono i posti e 1 riga. Indicano l'osservabilità e la controllabilità dei posti, quindi i vettori attesi dovrebbero avere tutti gli elementi pari a -1, essendo tutti i posti osservabili e controllabili. Però, in seguito all'esecuzione del *pluginP9* i vettori restituiti sono vuoti per tutte le reti testate;

   (ii) i vettori T_{obs} e T_{contr} entrambi di n colonne, tante quante sono le transizioni e 1 riga. Indicano l'osservabilità e la controllabilità delle transizioni. Il vettore T_{obs} atteso dovrebbe avere elementi pari a 1 in corrispondenza delle transizioni osservabili e pari a 0 in corrispondenza di quelle non osservabili. Il vettore T_{contr} invece dovrebbe avere tutti i suoi elementi pari a -1, essendo tutte le transizioni controllabili. Però, in seguito all'esecuzione del *pluginP9* i vettori restituiti sono vuoti per tutte le reti testate;

6. informazioni sulle transizioni, riportiamo solo il vettore di interesse per i nostri scopi

   (i) il vettore Mem di n colonne, tante quante sono le transizioni e 1 riga, i suoi elementi dovrebbero essere tutti pari a &. Però, in seguito all'esecuzione del *pluginP9* il vettore restituito è vuoto per tutte le reti provate.

5.2.6 Test numero due sugli adapters

In questa sezione verranno elencati i problemi incontrati nell'utilizzo degli *adapters* in seguito alla revisione da parte dell'ingegnere Contini di Akhela srl. Verranno riportati i problemi solo degli *adapter* sottoposti a revisioone. L'attenzione è ora puntata sui *adapter* utili per lo scopo della tesi.

L'*adapterA1.exe* che converte il formato Pnml_DISC nel Matlab_DISC ha restituito per le reti prese in considerazione un file .m in cui la matrice $Post$ è scorretta in corrispondenza delle transizioni non osservabili (comprese quelle di guasto), e dal vettore di celle F si deduce che non sono state riconosciute le diverse classi di guasto.

L'*adapterA9.exe* che converte il formato Matlab_DISC nel PN_Diag_Matlab funziona correttamente nelle RdP con una sola classe di guasto, ad esempio nella rete in Figura 5.9, mentre nelle reti in Figura 5.3 e 5.8 contenenti ciascuna più di una classe di guasto, non è stato possibile visualizzare il risultato, perchè l'*adapterA9* smette di funzionare.

5.2.7 Test numero tre sui plugin

In questa sezione non è più necessario elencare i problemi incontrati nell'utilizzo degli *plugin* in quanto questo terzo test ha restituito risultati corretti, grazie alla revisione effettuata dagli ingegneri Contini e Perria, sui *pluginP1*, *pluginP2*, *pluginP7* e *pluginP9*. Il test è stato eseguito su numerose RdP, aventi tutte caratteristiche diverse, tra cui quelle utilizzate nei test effettuati nelle sezioni precedenti, ovvero le RdP in Figura 5.3, 5.8 e 5.9. L'attenzione è ora puntata sui *plugin* utili per lo scopo della tesi.

Il *pluginP1.exe* che esegue la conversione della rete di Petri dal formato .xml al formato Pnml_DISC, pur funzionando correttamente già dal primo test effettuato, è stato migliorato, rendendo più comprensibile l'interfaccia Pnml Disc Editor. Con la modifica, l'elenco dei posti e delle transizioni viene riportato in ordine crescente, e non più in ordine alfabetico come in precedenza. E' opportuno però che la RdP di partenza, ovvero nel formato .xml di *Pipe3* parta dal posto p_1 e dalla transizione t_1, usando l'artificio spiegato nella sezione in cui si evidenziano i problemi riferiti al toolbox *Pipe3*. Questo è necessario per il funzionamento corretto dei toolbox *PN_DIAG* e *PN_DIAG_2*, che per come sono stati realizzati richiedono che la numerazione dei posti e delle transizioni parta dal numero uno. Si sta in ogni caso pensando ad un modo per evitare questo artificio poco elegante, e sarà compito dell'*adapterA9* che restituisce il file.m da mandare in ingresso ai toolbox sopra citati a doversi occupare di questo problema.

Il *pluginP2.exe* che coverte il formato Matlab_DISC nel formato Pnml_DISC, è stato corretto, ed ora tutte le transizioni vengono riconosciute dal formato .pnml, comprese perciò anche quelle non osservabili. Inoltre nella lista degli archi il posto p_0 e la transizione t_0 non vengono più inclusi, risulta perciò tutto coerente con l'elenco dei posti e delle transizioni.

IL *pluginP7.exe* che converte il grafo di raggiungibilità generato da *Tina* nel formato .fsm (di *Desuma*), è stato sottoposto a diverse modifiche. Iniazialmente ad un primo test sembrava funzionare perfettamente, perchè mettendo a confronto il grafo di raggiungibilità generato da *Tina* con quello restituito in seguito all'esecuzione del *pluginP7* si aveva una corrispondenza perfetta. Osservando con più

attenzione, ci siamo resi conto di certi particolari che non permettevano il corretto funzionamento degli eseguibili *diag_UR* e *dcycle*, contenuti nella libreria UMDES. Per capire meglio, facciamo un esempio, consideriamo la Rete di Petri rappresentata in Figura 5.8. Il grafo di raggiungibilità per questa rete generato da *Tina* si può osservare in Figura 5.13, dove lo stato 0 corrisponde ad una marcatura nel posto p_1, lo stato 1 ad una marcatura nel posto p_2, lo stato 2 ad una marcatura nel posto p_3, lo stato 3 ad una marcatura nel posto p_4, lo stato 4 ad una marcatura nel posto p_5, lo stato 5 ad una marcatura nel posto p_6 e infine lo stato 6 ad una marcatura nel posto p_7. Inoltre conosciamo l'etichetta corrispondente a ciascuna transizione, ovvero sappiamo che a t_1 corrisponde l'etichetta a, a t_2 e t_3 l'etichetta b, a t_4 l'etichetta c e tutte le altre sono non osservabili, si riconoscono dalla E prima del nome, inoltre la F indica che la transizione è di guasto. Dal grafo di raggiungibilità utilizzando il *pluginP7* si ottiene l'automa in Figura 5.14, che apparentemente sembra corretto, ma in realtà non fornisce le informazioni sulle etichette associate alle transizioni, quindi in presenza di transizioni con uguale etichetta il diagnosticatore creato dall'eseguibile *diag_UR* a partire da questo automa non è quello atteso, anche se comunque riconosce quali sono le transizioni osservabili (indicate con o) e non osservabili (indicate con uo). Il diagnosticatore prodotto si può osservare in Figura 5.15. L'automa atteso è mostrato in Figura 5.16 e il corrispondente diagnosticatore si può vedere in Figura 5.17. Facendo riferimento alla rete in Figura 5.3, o ad una qualsiasi rete con un numero di transizioni maggiore di nove, si ha un ulteriore problema, non distingue più quali sono le transizioni osservabili e quelle non osservabili. Dopo aver fatto notare questi problemi agli sviluppatori del convertitore, sono stati eliminati tutti i bachi che impedivano il corretto funzionamento della conversione. Ora il *pluginP7* fornisce l'automa atteso con tutte le informazioni necessarie.

Il *pluginP9.exe* permette la conversione dal formato PN_Diag_Matlab al formato Matlab_DISC, è stato corretto, tutti i vettori che nei test precedenti erano vuoti, ora con i nuovi test effettuati su varie reti di Petri, con caratteristiche diverse le une dalle altre, contengono tutte le informazioni necessarie per poter avere un file in formato Matlab_DISC corretto. Tuttavia, inizialmente si aveva ancora un problema da risolvere. I test sono stati eseguiti su una macchina avente come sistema operativo Windows 7 a 64 bit, e il *pluginP9* funzionava solo sotto certe condizioni, in particolare solo se veniva posizionato nel desktop della macchina, in qualsiasi altra posizione, al momento dell'esecuzione la conversione si interrompeva, mentre gli stessi test eseguiti su una macchina con Windows 7 a 32 non provocavano nessun tipo di problema. Si è testato anche su una macchina avente come sistema operativo Windows Vista a 32 bit, ma si verificavano gli stessi problemi avuti nel Windows 7 a 64 bit. Si è iniziato a sospettare che ci fosse un problema di compatibilità fra i vari sistemi operativi. Abbiamo allora

```
INPUT NET --------------------

parsed net noname

7 places, 8 transitions

net noname
tr t1 : a p1 -> p2
tr t2 : b p2 -> p3
tr t3 : b p6 -> p7
tr t4 : c p7 -> p7
tr t5 : F5 p3 -> p4
tr t6 : E6 p4 -> p5
tr t7 : F7 p5 -> p2
tr t8 : F8 p4 -> p6
pl p1 : P1 (1)
pl p2 : P2
pl p3 : P3
pl p4 : P4
pl p5 : P5
pl p6 : P6
pl p7 : P7
```

```
REACHABILITY ANALYSIS -----------

bounded

7 marking(s), 8 transition(s)

MARKINGS:

0 : p1
1 : p2
2 : p3
3 : p4
4 : p5
5 : p6
6 : p7

REACHABILITY GRAPH:

0 -> t1/1
1 -> t2/2
2 -> t5/3
3 -> t6/4, t8/5
4 -> t7/1
5 -> t3/6
6 -> t4/6
```

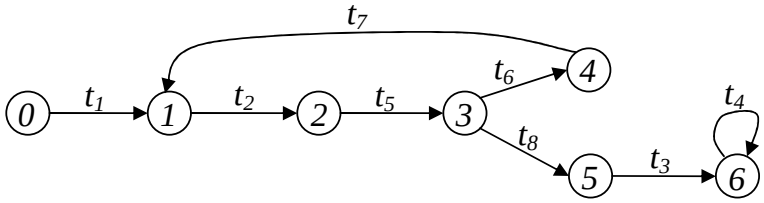

Figura 5.13: Grafo di raggiungibilità generato dal toolbox *Tina* per la rete in Figura 5.8

testato il *pluginP9* su una macchina avente come sistema operativo Windows Xp, i risultati forniti erano tutti in accordo con quelli attesi. Sono stati fatti dei test anche su altre macchine con Windows 7 a 64 bit con la sorpresa che nessun errore si manifestava. Quindi il problema continua persisteva solo sul mio personal computer. Nel frattempo che gli sviluppatori pensavano a come risolvere questo problema,si è potuto usare il *pluginP9* anche sul mio personale computer, dotato di Windows 7 a 64 bit, però modificando le proprietà dell'eseguibile facendo in modo che funzionasse in modalità di compatibilità con Windows Xp. In ogni caso dopo molto lavoro, gli sviluppatori sono riusciti a risolvere il problema, sia per Windows Vista, sia per il mio personal computer. E' stato necessario compilare l'eseguibile su una macchina a 64 bit per far si che il *pluginP9* funzionasse anche sul mio calcolatore.

```
7

S0      0       1
t1      S1      c       o

S1      0       1
t2      S2      c       o

S2      0       1
t5      S3      c       uo

S3      0       2
t6      S4      c       uo
t8      S5      c       uo

S4      0       1
t7      S1      c       uo

S5      0       1
t3      S6      c       o

S6      0       1
t4      S6      c       o
```

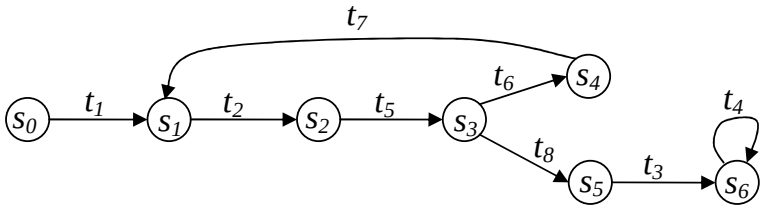

Figura 5.14: Automa apparentemente corretto in riferimento alla Figura 5.8 ottenuto utilizzando il *pluginP7*

5.2.8 Test numero tre sugli adapters

In questa sezione non è più necessario elencare i problemi incontrati nell'utilizzo degli *adapters* in quanto questo terzo test ha restituito risultati corretti, grazie alla revisione effettuata dagli ingegneri Contini e Perria, sugli *adapter1* e *adapterA9*. Il test è stato eseguito su numerose reti di Petri, aventi tutte caratteristiche diverse, tra cui quelle utilizzate nei test effettuati nelle sezioni precedenti, ovvero le reti di Petri in Figura 5.3, 5.8 e 5.9. L'attenzione è puntata sugli *adapter* utili per lo scopo della tesi.

L'*adapterA1.exe* che converte la rete di Petri dal formato Pnml_DISC al formato Matlab_DISC, con le modifiche apportate restituisce un file.m aventi tutte

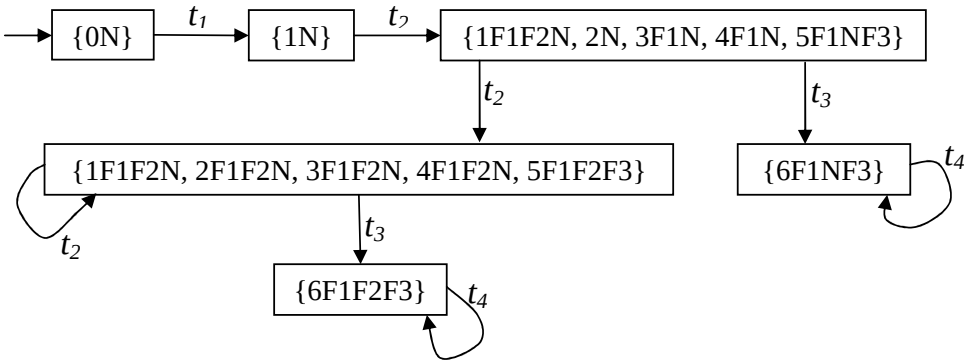

Figura 5.15: Diagnosticatore errato generato per l'automa in Figura 5.14 in seguito all'utilizzo del *pluginP7*

le informazioni corrette, in particolare, ora la matrice $Post$ è corretta in corrispondenza di tutte le transizioni, siano esse osservabili oppure no. Invece, per quanto riguarda il vettore di celle F ancora non è stato trovato un modo per far si che in presenza di più classi di guasto fornisca il risultato atteso. Infatti, per la RdP in Figura 5.3, avendo la rete 13 transizioni, il vettore restituito è il seguente $F = [-1;-1;-1;-1;-1;-1;-1;-1;-1;-1;1;1;-1]$, dove il -1 corrisponde ad una transizione non di guasto, mentre l'1 indica che la transizione corrispondente è di guasto e appartiene alla classe di guasto 1, in questo caso sono le transizioni non osservabili ε_{11} e ε_{12}. Il vettore atteso era il seguente $F = [-1;-1;-1;-1;-1;-1;-1;-1;-1;-1;1;2;-1]$, dove l'unica differenza è il 2 in corrispondenza della transizione di guasto ε_{12}. In questo modo la transizione di guasto $\varepsilon_{11} \in T_f^1$ e la $\varepsilon_{12} \in T_f^2$, come dalle specifiche. Poichè subito dopo l'*adapteA1.exe* viene eseguito l'*adapterA9*, prima di mandare in esecuzione quest'ultimo modifichiamo manualmente il file.m restituito dall'*adapterA1* solo in corrispondenze delle vettore F, ovvero modifichiamo solo i numeri in corrispondenze delle transizioni appartenenti a classi di guasto diverse dalla prima. Quindi per la rete in Figura 5.3 il vettore F è stato modificato nel seguente modo $F = [-1;-1;-1;-1;-1;-1;-1;-1;-1;-1;1;2;-1]$, e per la rete in Figura 5.8, costituita da tre diverse classi di guasto, il vettore F è stato modificato nel seguente modo $F = [-1;-1;-1;-1;1;-1;2;3]$, dove 1 corrisponde alla transizione di guasto $\varepsilon_5 \in T_f^1$, 2 alla transizione di guasto $\varepsilon_7 \in T_f^2$ e 3 corrisponde alla transizione di guasto $\varepsilon_8 \in T_f^3$, come dalle specifiche.

```
7

S0      0       1
a       S1      c       o

S1      0       1
b       S2      c       o

S2      0       1
F5      S3      c       uo

S3      0       2
E6      S4      c       uo
F8      S5      c       uo

S4      0       1
F7      S1      c       uo

S5      0       1
b       S6      c       o

S6      0       1
c       S6      c       o
```

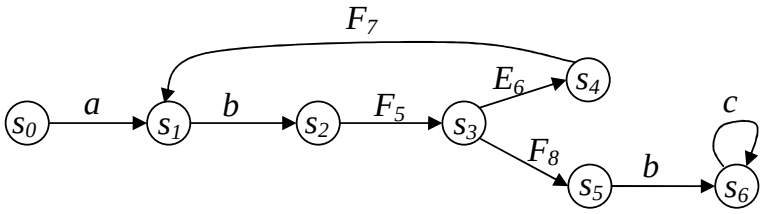

Figura 5.16: Automa atteso per la rete in Figura 5.8

L'*adapterA9.exe* che esegue la conversione dal formato Matlab_DISC al formato PN_Diag_Matlab funziona perfettamente. Bisogna ricordarsi però che in presenza di un numero di classi di guasto superiore a uno, prima di eseguire la conversione con l'*adapterA9* il file.m restituito dall'*adapterA1* deve essere modificato manualmente in corrispondenza delle transizioni di guasto appartenenti a classi di guasti superiori alla uno, in questo modo il vettore F, per esempio, per la rete di Petri in Figura 5.3, è $F = \{[11];[12]\}$, ovvero riconosce la presenza di due classi di guasto, per la rete in Figura 5.8 è $F = \{[5];[7];[8]\}$, riconoscendo perfettamente le tre classi di guasto. In ogni caso gli sviluppatori del convertitore stanno pensando ad un modo per automatizzare questa operazione.

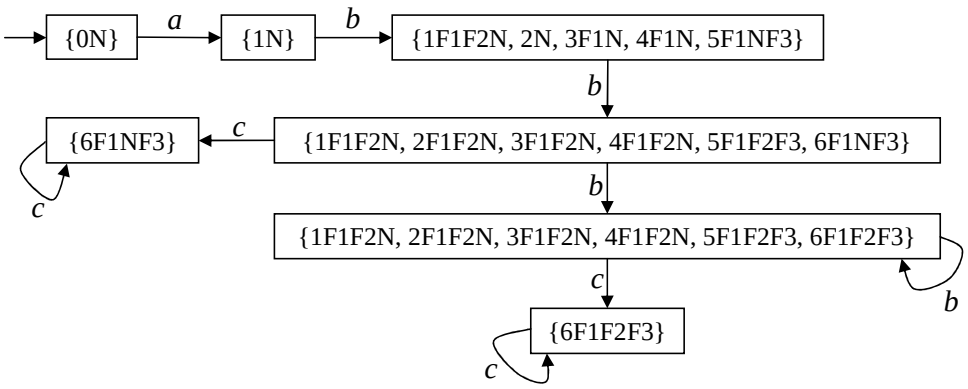

Figura 5.17: Diagnosticatore giusto generato per l'automa in Figura 5.14

Capitolo 6

Analisi dei risultati numerici

Sommario

In questo capitolo riportiamo alcuni esempi di reti di Petri di cui abbiamo studiato la diagnosticabilità utilizzando i tool *PN_DIAG*, *PN_DIAG_2* e *UMDES*. Il tutto è stato fatto utilizzando la *Piattaforma Software*. Riportiamo per chiarezza d'esposizione anche le figure rappresentanti le reti di Petri e le strade seguite nell'utilizzo della piattaforma. Viene inoltre preso in considerazione un modello fisico parametrico, utilizzato per fare un confronto tra i diversi approcci per la verifica della diagnosticabilità. Sono state fatte delle simulazioni per diversi valori dei parametri e raccolti i risultati ottenuti facendo le dovute considerazioni.

6.1 Primo modo di utilizzo della Piattaforma Software

Poichè lo scopo della tesi è mettere a confronto i diversi approcci di diagnosi basati sia su automi che su reti di Petri. In questo modo, la verifica della diagnosticabilità della rete di Petri data si può effettuare attraverso i tool *PN_DIAG* e *PN_DIAG_2*. Dalla rete di Petri si passa all'automa ad essa corrispondente e tramite gli eseguibili contenuti della libreria *UMDES* si può verificare la diagnosticabilità. I risultati ottenuti dai tre tool si mettono a confronto per evidenziare i vantaggi di un tool rispetto ad un altro. Gli esempi che seguono mostrano uno dei modi per utilizzare la *Piattaforma Software*. Riportiamo in Figura 6.1 la strada seguita. Per poter utilizzare la *Piattaforma Software* è utile creare uno script (scritto in linguaggio Matlab) nella sezione *Script Manager* in cui vengono scrit-

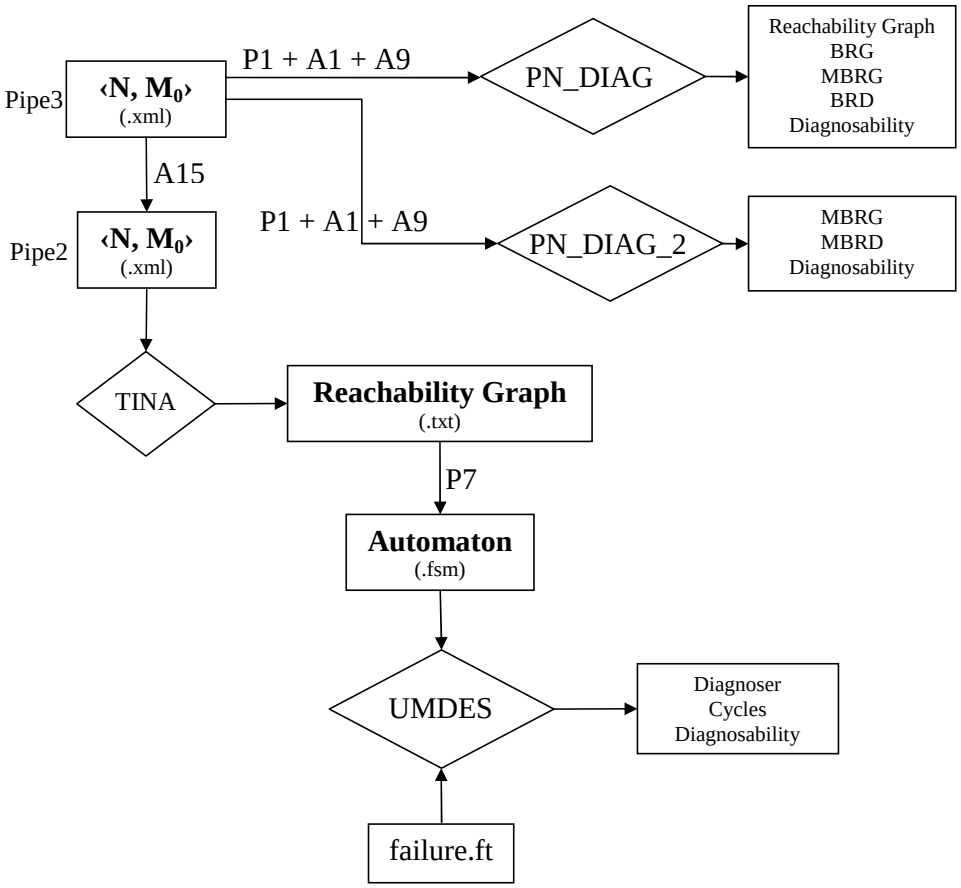

Figura 6.1: Primo modo di utilizzo della *Piattaforma Software*

te le operazioni che si intendono eseguire. Una volta mandato in esecuzione lo script, nell'output console della piattaforma vengono riassunti i risultati ottenuti, che possono essere copiati in un file apposito, in modo da tenere traccia delle informazioni date e non dover rilanciare lo script.

6.1.1 Esempi a partire da una RdP disegnata con Pipe3

Abbiamo già spiegato nel Capitolo 5 lo schema riassunto in Figura 6.1. Consideriamo la rete di Petri in Figura 6.2, in particolare iniziamo con il considerare il caso in cui l'insieme delle transizioni osservabili è $T_o = \{t_1, t_2, t_3, t_4\}$, l'insieme delle transizioni non osservabili è $T_u = \{t_5, t_6\}$, le classi di guasto sono $T_f = T_f^1 \cup T_f^2 = \{t_5\} \cup \{t_6\}$. La funzione di etichettatura è tale che: $\mathscr{L}(t_1) = a$, $\mathscr{L}(t_2) = b$, $\mathscr{L}(t_3) = c$ e $\mathscr{L}(t_4) = d$.

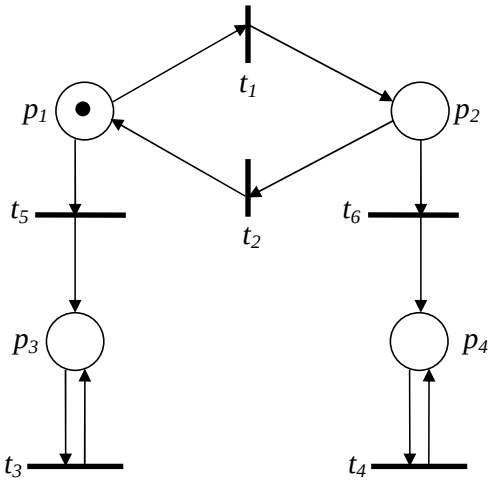

Figura 6.2: Esempio di rete di Petri generica

Per prima cosa abbiamo disegnato la rete di Petri, tramite *Pipe3* e abbiamo chiamato il file rete3_1_p3.xml. Poi abbiamo scritto due script (riportati in Appendice A.2) all'interno della piattaforma, chiamati script_desuma nell'Appendice A.2.1 e script_tools nell'Appendice A.2.2, entrambi con lo scopo di stabilire la diagnosticabilità della rete. Nel primo vengono eseguite le conversioni tramite l'*adapterA15_Pipe3toPipe2*, *Tina*, *pluginP7*, *pluginP1* e l'*adapterA1*, che forniscono nell'ordine i seguenti file:

1. rete3_1_p2.xml, file generato dall'*adapterA15_Pipe3toPipe2*, cioè dato in ingresso il file rete3_1_p3.xml (formato compatibile con il tool *Pipe3*) restituisce il file compatibile con il tool *Pipe2* che rappresenta la rete disegnata in partenza;

2. reach_graph_1.txt, file generato dal tool *Tina*, grazie all'esecuzione del comando $tina.exe -R -PNML\ rete3\_1\_p2.xml\ reach\_graph\_1.txt$, dove in ingresso si ha la rete nel formato compatibile con il tool *Pipe2* e in uscita si ha il grafo di raggiungibilità della rete data in ingresso;

3. reach_graph_1.fsm, file generato in seguito all'esecuzione del *pluginP7*, cioè dato in ingresso il grafo di raggiungibilità generato da *Tina* in uscita si ha l'automa ad esso corrispondente nel formato di *Desuma* (.fsm);

4. rete3_1_pnml.pnml, file generato in seguito all'esecuzione dell'*pluginP1*, cioè dato in ingresso il file rete3_1_p3.xml restituisce il file nel formato

Pnml_DISC, che rappresenta la rete di Petri compatibile con il Pnml Disc Editor, dove si possono avere tutte le informazioni sulla rete in considerazione;

5. rete3_1_matlab_disc.m, file generato dall'esecuzione dell'*adapterA1*, cioè dato in ingresso il file nel formato Pnml_DISC in uscita si ha il corrispondente file nel formato Matlab_DISC, che contiene le informazioni sulla rete considerata.

Nello stesso script sono stati inseriti gli eseguibili *diag_UR* e *dcycle* così da avere il diagnosticatore e il numero di cicli indeterminati presenti nel sistema in considerazione, chiamati diagnoser_1.diag (e diagnoser_1.fsm) e inputFile_1.cycles. Per poter tenere traccia dei tempi impiegati per la simulazione, sono stati inseriti dei contatori, facendo in modo che i valori numerici venissero memorizzati nel file Desuma_3.txt, in questo file vengono trascritti i seguenti tempi:

1. TIME_R, tempo espresso in secondi per il calcolo dell'automa di cui si vuole studiare la diagnosticabilità;
2. TIME_DIAG, tempo espresso in secondi per il calcolo del diagnosticatore;
3. TIME_CICLI, tempo espresso in secondi per il calcolo del numero di cicli indeterminati;
4. TIME_UMDES, tempo espresso in secondi per avere informazioni sulla diagnosticabilità del sistema, non è altro che la somma dei tempi sopra citati.

Nel secondo script (riportato in Appendice A.2.2) viene eseguita la conversione tramite l'*adapterA9* che restituisce il file: rete3_1_input.m, in ingresso viene dato il file rete3_1_matlab_disc.m, il file generato caratterizza la rete di Petri attraverso $Pre, Post, M0, F, L, E$, ovvero rappresenta il file da mandare in ingresso ai toolbox di Pocci e Perria. Inoltre vengono forniti i comandi per mandare in esecuzione i toolbox *PN_DIAG* e *PN_DIAG_2*, anche per questi abbiamo tenuto traccia dei tempi e della cardinalità dei grafi costruiti durante la simulazione nei file:

- PN_DIAG_3.txt, che memorizza:
    1. $|R|$, cardinalità del grafo di raggiungibilità;

2. TIME_R, tempo espresso in secondi per il calcolo del grafo di raggiungibilità;
3. $|brg|$, cardinalità del *Basis Reachability Graph* (BRG);
4. TIME_brg, tempo espresso in secondi per il calcolo del BRG;
5. $|mbrg|$, cardinalità del *Modified Basis Reachability Graph* (MBRG);
6. TIME_mbrg, tempo espresso in secondi per il calcolo del MBRG;
7. $|brd|$, cardinalità del *Basis Reachability Diagnoser* (BRD);
8. TIME_brd, tempo espresso in secondi per il calcolo del BRD;
9. TIME_diagnosability, tempo espresso in secondi per determinare se il sistema è diagnosticabile oppure no.

- PN_DIAG_2_3.txt, che memorizza:

1. $|mbrg|$, cardinalità del *Modified Basis Reachability Graph* (MBRG);
2. TIME_mbrg, tempo espresso in secondi per il calcolo del MBRG;
3. $|mbrd|$, cardinalità del *Modified Basis Reachability Graph* (MBRD);
4. TIME_mbrd, tempo espresso in secondi per il calcolo del MBRD;
5. TIME_diagnosability, tempo espresso in secondi per determinare se il sistema è diagnosticabile oppure no.

E' stato necessario creare due script (entrambi riportati in Appendice A.2) perchè l'*adapterA9* funziona perfettamente solo se prima di eseguire la conversione il file_matlab_disc.m restituito dall'*adapterA1* viene modificato manualmente, in corrispondenza del vettore di celle F, che tiene traccia delle transizioni di guasto. La modifica manuale si deve fare solo nel caso in cui le classi di guasto siano in numero superiori a uno, in caso contrario sarebbe stato possibile anche creare un solo script racchiudente tutti i toolbox da mettere a confronto. Quindi una volta eseguito lo script_desuma si modifica il file rete3_1_matlab_disc.m in modo che l'*adapterA9* restituisca il file rete3_1_input.m atteso. Utilizzando la rete in Figura 6.2, si è pensato di testare i toolbox modificando l'insieme delle transizioni osservabili, non osservabili e/o di guasto, così da poter confrontare quale dei tre toolbox sia più efficiente sia in termini di tempo, sia per quanto riguarda la cardinalità dello spazio di stato dei grafi, così da renderci conto come la cardinalità possa crescere/diminuire a seconda dell'approccio proposto. Abbiamo considerato in tutto sei reti etichettate, che si ottengono dalla rete posto/transizione in Figura 6.2 con la funzione di etichettatura data in Tabella 6.1. Le sei reti di Petri sono state chiamate:

$RETE$	T_o	T_u	T_f	$\mathscr{L}$
RETE 3_1	$T_o = \{t_1, t_2, t_3, t_4\}$	$T_u = \{t_5, t_6\}$	$T_f^1 = \{t_5\}$ $T_f^2 = \{t_6\}$	$\mathscr{L}(t_1) = a$ $\mathscr{L}(t_2) = b$ $\mathscr{L}(t_3) = c$ $\mathscr{L}(t_4) = d$
RETE 3_2	$T_o = \{t_1, t_2, t_3, t_4\}$	$T_u = \{t_5, t_6\}$	$T_f = \{t_5, t_6\}$	$\mathscr{L}(t_1) = a$ $\mathscr{L}(t_2) = b$ $\mathscr{L}(t_3) = c$ $\mathscr{L}(t_4) = d$
RETE 3_3	$T_o = \{t_1, t_3, t_4, t_6\}$	$T_u = \{t_2, t_5\}$	$T_f = \{t_2, t_5\}$	$\mathscr{L}(t_1) = a$ $\mathscr{L}(t_3) = c$ $\mathscr{L}(t_4) = d$ $\mathscr{L}(t_6) = b$
RETE 3_4	$T_o = \{t_2, t_3, t_4, t_6\}$	$T_u = \{t_1, t_5\}$	$T_f = \{t_1, t_5\}$	$\mathscr{L}(t_2) = b$ $\mathscr{L}(t_3) = c$ $\mathscr{L}(t_4) = d$ $\mathscr{L}(t_6) = a$
RETE 3_5	$T_o = \{t_1, t_3, t_4, t_6\}$	$T_u = \{t_2, t_5\}$	$T_f^1 = \{t_2\}$ $T_f^2 = \{t_5\}$	$\mathscr{L}(t_1) = a$ $\mathscr{L}(t_3) = c$ $\mathscr{L}(t_4) = d$ $\mathscr{L}(t_6) = b$
RETE 3_6	$T_o = \{t_2, t_3, t_4, t_6\}$	$T_u = \{t_1, t_5\}$	$T_f^1 = \{t_1\}$ $T_f^2 = \{t_5\}$	$\mathscr{L}(t_2) = b$ $\mathscr{L}(t_3) = c$ $\mathscr{L}(t_4) = d$ $\mathscr{L}(t_6) = a$

Tabella 6.1: Riepilogo dei casi considerati utilizzando la rete di Petri in Figura 6.2

1. rete3_1_p3.xml;
2. rete3_2_p3.xml;
3. rete3_3_p3.xml;
4. rete3_4_p3.xml;
5. rete3_5_p3.xml;
6. rete3_6_p3.xml.

Nella Tabella 6.2 riportiamo i risultati ottenuti per le reti prese in considerazione, verificando la diagnosticabilità delle reti attraverso i tre toolbox. Lo

script_desuma, riportato in Appendice A.2.1, permette di studiare la diagnosticabilità, grazie ai convertitori e agli eseguibili *diag_UR* e *dcycle*. Tramite lo script_tools, riportato in Appendice A.2.2, studiamo la diagnosticabilità utilizzando i software *PN_DIAG* e *PN_DIAG_2*.

La Tabella 6.2 è divisa in tre parti, la prima si riferisce agli eseguibili di *UMDES*, la seconda al toolbox *PN_DIAG* e la terza al toolbox *PN_DIAG_2*.

La parte relativa a *UMDES* è strutturata nel seguente modo:

- la colonna 1 indica la rete che stiamo considerando ($RETE$);
- la colonna 2 indica il numero di nodi dell'automa a stati finiti ($\#\ nodi$);
- la colonna 3 indica il tempo di calcolo espresso in secondi per generare l'automa ($t_R[s]$);
- la colonna 4 indica il numero di stati del diagnosticatore ($\#\ stati\ diag$);
- la colonna 5 indica il tempo espresso in secondi per il calcolo del diagnosticatore ($t_{diag}[s]$);
- la colonna 6 indica il numero di cicli indeterminati trovati tramite *dcycle* ($\#\ cicli\ indet$);
- la colonna 7 indica il tempo di calcolo espresso in secondi per la ricerca di cicli indeterminati ($t_{cicli}[s]$);
- la colonna 8 indica il tempo totale espresso in secondi per la verifica della diagnosticabilità ($t_{UMDES}[s]$);
- la colonna 9 indica se il sistema è diagnosticabile oppure no.

La parte relativa al software *PN_DIAG* è strutturata nel seguente modo:

- la colonna 1 indica la rete di Petri presa in considerazione ($RETE$);
- le colonne 2 e 3 indicano la cardinalità del grafo di raggiungibilità ($|R|$) e il tempo di calcolo espresso in secondi ($t_R[s]$), rispettivamente;
- le colonne 4 e 5 indicano la cardinalità del BRG ($|BRG|$) e il tempo di calcolo espresso in secondi ($t_{BRG}[s]$), rispettivamente;

- le colonne 6 e 7 indicano la cardinalità dell'MBRG ($|MBRG|$) e il tempo di calcolo espresso in secondi ($t_{MBRG}[s]$), rispettivamente;
- le colonne 8 e 9 indicano la cardinalità del BRD ($|BRD|$) e il tempo di calcolo espresso in secondi ($t_{BRD}[s]$), rispettivamente;
- la colonna 10 indica il tempo di calcolo per la verifica della diagnosticabilità espresso in secondi ($t_{diag}[s]$);
- la colonna 11 indica se il sistema è diagnosticabile oppure no.

La parte relativa al software *PN_DIAG_2* presenta la seguente struttura:

- la colonna 1 indica la rete di Petri considerata ($RETE$);
- le colonne 2 e 3 indicano la cardinalità dell'MBRG ($|MBRG|$) e il tempo di calcolo espresso in secondi ($t_{MBRG}[s]$), rispettivamente;
- le colonne 4 e 5 indicano la cardinalità dell'MBRD ($|MBRD|$) e il tempo di calcolo espresso in secondi ($t_{MBRD}[s]$), rispettivamente;
- la colonna 6 indica la classe di guasto a cui si sta facendo riferimento ($classe\ guasto$);
- la colonna 7 indica il numero di cicli incerti trovati all'interno dell'MBRD ($\#\ c.\ incerti$), ovvero il numero di cicli incerti in riferimento ad ogni classe di guasto considerata;
- la colonna 8 indica il tempo di calcolo per la verifica della diagnosticabilità espresso in secondi ($t_{diag}[s]$);
- la colonna 9 indica se il sistema è diagnosticabile oppure no.

Osservando i risultati riportati nella Tabella 6.2 non si evidenzia nessuna differenza importante tra i diversi approcci di diagnosi. Infatti i tempi, espressi in secondi, per verificare la diagnosticabilità sono brevi per tutti i diversi tipi di approccio. La cardinalità dello spazio di stato del diagnosticatore e del MBRD è identica. Quindi possiamo dire che per reti di piccole dimensioni, dove il numero di transizioni non osservabili non è molto maggiore del numero di transizioni osservabili, e il numero di stati del grafo di raggiungibilità (o equivalentemente del numero di nodi dell'automa) non è eccessivamente grande i diversi approcci sono pressochè equivalenti.

UMDES								
RETE	*# nodi*	$t_R[s]$	*# stati diag*	$t_{diag}[s]$	*# cicli indet*	$t_{cicli}[s]$	$t_{UMDES}[s]$	*Diag?*
3_1	4	0.1502	4	0.1088	0	0.0617	0.0621	si
3_2	4	0.0587	4	0.0660	0	0.0558	0.0561	si
3_3	4	0.0635	6	0.0824	0	0.0524	0.0529	si
3_4	4	0.0580	4	0.0800	0	0.0611	0.0614	si
3_5	4	0.0642	7	0.1137	0	0.0519	0.0524	si
3_6	4	0.0597	5	0.0801	0	0.0737	0.0741	si

PN_DIAG										
RETE	$\|R\|$	$t_R[s]$	$\|BRG\|$	$t_{BRG}[s]$	$\|MBRG\|$	$t_{MBRG}[s]$	$\|BRD\|$	$t_{BRD}[s]$	$t_{diag}[s]$	*Diag?*
3_1	4	0.0372	4	0.0914	4	0.0370	5	0.1503	0.2134	si
3_2	4	0.00057	4	0.0104	4	0.0125	5	0.0165	0.1046	si
3_3	4	0.00055	4	0.0112	4	0.0118	7	0.0202	0.0330	si
3_4	4	0.00056	3	0.0086	4	0.0117	5	0.0148	0.0271	si
3_5	4	0.00056	4	0.0111	4	0.01243	7	0.0206	0.0554	si
3_6	4	0.0006	3	0.0087	4	0.0122	5	0.0153	0.0470	si

PN_DIAG_2								
RETE	$\|MBRG\|$	$t_{MBRG}[s]$	$\|MBRD\|$	$t_{MBRD}[s]$	*classe guasto*	*# c. incerti*	$t_{diag}[s]$	*Diag?*
3_1	4	0.1218	4	0.0800	una due	0 0	0.2475	si
3_2	4	0.0156	4	0.0187	una	1	0.0857	si
3_3	4	0.0129	6	0.0220	una	0	0.0402	si
3_4	4	0.0123	4	0.0167	una	0	0.0306	si
3_5	4	0.0327	7	0.0228	una due	0 1	0.0597	si
3_6	4	0.0127	5	0.0190	una due	0 1	0.0358	si

Tabella 6.2: Risultati della simulazione effettuata con i toolbox *UMDES*, *PN_DIAG*, *PN_DIAG_2* prendendo in considerazione la rete in Figura 6.2

6.2 Secondo modo di utilizzo della Piattaforma Software

Gli esempi che seguono mostrano un secondo modo per utilizzare la *Piattaforma Software*. Riportiamo in Figura 6.3 la strada seguita. Lo scopo è sempre quello di mettere a confronto i diversi approcci per la verifica della diagnosticabilità, quindi vedere come varia la cardinalità dello spazio di stato dei diversi grafi creati all'interno dei toolbox di Pocci e Perria rispetto a quella che si ha utilizzando l'approccio riferito agli automi. Per poter utilizzare la *Piattaforma Software* è utile creare uno script (scritto in linguaggio Matlab) nella sezione *Script Manager* in cui vengono scritte le operazioni che si intendono eseguire. Una volta mandato in esecuzione lo script, nell'output console della piattaforma vengono riassunti i risultati ottenuti, che possono essere copiati in un file appo-

sito, in modo da tenere traccia delle informazioni date e non dover rilanciare lo script. Questo secondo modo di utilizzo è più interessante, in quanto permette di verificare la diagnosticabilità anche di reti molto grandi, o che variano al variare di certi parametri, impossibili da disegnare tramite *Pipe3*. Vedremo per prima cosa alcuni esempi molto semplici che permettono di capire in che modo opera la piattaforma.

6.2.1 Esempi a partire da una RdP nel formato PN_DIAG

Abbiamo già spiegato nel Capitolo 5 lo schema utilizzato, riassunto in Figura 6.3. Consideriamo la rete di Petri in Figura 6.4 in cui l'insieme delle transizioni osservabili è $T_o = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$, l'insieme delle transizioni non osservabili è $T_u = \{\varepsilon_8, \varepsilon_9, \varepsilon_{10}, \varepsilon_{11}, \varepsilon_{12}, \varepsilon_{13}\}$. Sono presenti due classi di guasto $T_f = T_f^1 \cup T_f^2 = \{\varepsilon_{11}\} \cup \{\varepsilon_{12}\}$. La funzione di etichettatura è tale che: $\mathscr{L}(t_1) = a$, $\mathscr{L}(t_2) = \mathscr{L}(t_3) = b$, $\mathscr{L}(t_4) = \mathscr{L}(t_5) = c$ e $\mathscr{L}(t_6) = \mathscr{L}(t_7) = d$. Il numero di gettoni nel posto p_1 varia da 1 a 3, quindi per poter partire da una rete disegnata con *Pipe3*, sarebbe necessario disegnare tre RdP, ovvero tante quanto il numero di variazione dei gettoni. Fino a quando le reti da disegnare sono poche si può seguire il primo modo di utilizzo spiegato precedentemente, ma è raro dovere avere a che fare con modelli piccoli e semplici, dove varia un solo parametro. Quindi è molto utile seguire il secondo schema (Figura 6.3). Per prima cosa è stato creato il file $rete22\_input.m$, dove sono state riportate la matrice Pre e $Post$, il vettore della marcatura iniziale $M0$, la matrice di celle F (contenente le transizioni di guasto), la matrice di celle L (che indica l'indice delle transizioni etichettate), la matrice di celle E (che indica le etichette associate alla transizioni) ed infine viene definito un vettore net_array che racchiude tutte le informazioni precedenti ($net\_array = \{Pre, Post, M0, F, L, E\}$). Una volta che tale file è a disposizione è possibile mandarlo in ingresso ai toolbox *PN_DIAG* e *PN_DIAG_2*, quindi possiamo avere subito le informazioni circa la diagnosticabilità della rete. Per rendere le cose più veloci si è scritto lo script, chiamato script_tools riportato in Appendice A.3.2, in cui vengono richiamati i toolbox sopra citati, e si tiene traccia dei tempi e della cardinalità dei grafi costruiti durante la simulazione nei file:

- risultati_22_PN_DIAG.txt, che memorizza:
  1. $|R|$, cardinalità del grafo di raggiungibilità;
  2. TIME_R, tempo espresso in secondi per il calcolo del grafo di raggiungibilità;
  3. $|brg|$, cardinalità del *Basis Reachability Graph* (BRG);

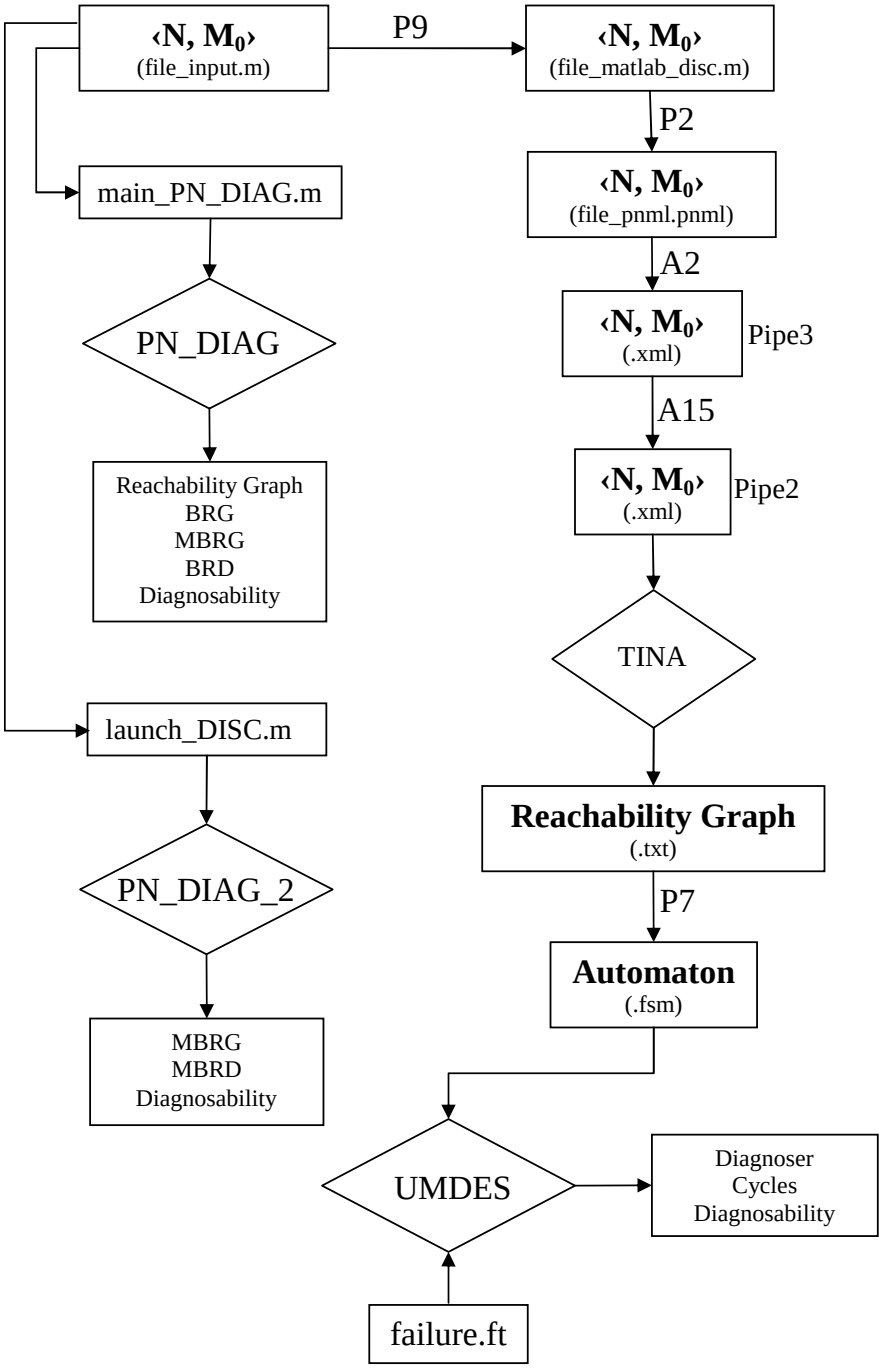

Figura 6.3: Secondo modo di utilizzo della *Piattaforma Software*

4. TIME_brg, tempo espresso in secondi per il calcolo del BRG;
5. $|mbrg|$, cardinalità del *Modified Basis Reachability Graph* (MBRG);
6. TIME_mbrg, tempo espresso in secondi per il calcolo del MBRG;
7. $|brd|$, cardinalità del *Basis Reachability Diagnoser* (BRD);
8. TIME_brd, tempo espresso in secondi per il calcolo del BRD;
9. TIME_diagnosability, tempo espresso in secondi per determinare se il sistema è diagnosticabile oppure no.

- risultati_22_PN_DIAG_2.txt, che memorizza:

1. $|mbrg|$, cardinalità del *Modified Basis Reachability Graph* (MBRG);
2. TIME_mbrg, tempo espresso in secondi per il calcolo del MBRG;
3. $|mbrd|$, cardinalità del *Modified Basis Reachability Graph* (MBRD);
4. TIME_mbrd, tempo espresso in secondi per il calcolo del MBRD;
5. TIME_diagnosability, tempo espresso in secondi per determinare se il sistema è diagnosticabile oppure no.

Nell'output console della piattaforma viene indicato se la rete è o meno diagnosticabile. Inoltre per il toolbox di *PN_DIAG_2* viene riportato il numero di cicli incerti presenti nelle classi di guasto. Per poter fare il confronto con i risultati forniti da *UMDES* è strettamente necessario fare uso della piattaforma. In particolare, si è scritto uno script, chiamato script_desuma riportato in Appendice A.3.1, che permette di creare tante reti al variare del parametro presente (in questo caso il parametro è x ed equivale al numero di gettoni presenti nel posto p_1). Per ogni rete creata si utilizzano nell'ordine il *pluginP9*, *pluginP2*, l'*adapterA2*, *adapterA15_Pipe3toPipe2*, *Tina*, il *pluginP7*, che restituiscono i seguenti file:

1. rete1_matlab_disc.m, rete2_matlab_disc.m e rete1_matlab_disc.m; tutti e tre i file sono nel formato Matlab_DISC, per poter essere creati in ingresso al *pluginP9* viene mandato il file nel formato PN_DIAG_MATLAB;

2. rete1_pnml.pnml, rete2_pnml.pnml, rete3_pnml.pnml; tutti e tre i file restituiti sono nel formato Pnml_DISC, creati dal *pluginP2* dandogli in ingresso i file in formato Matlab_DISC, si può verificare la correttezza di questi file, aprendoli dal Pnml Disc Editor, che riassume le caratteristiche delle reti;

3. rete1_p3.xml, rete2_p3.xml, rete3_p3.xml; tutti e tre i file restituiti sono nel formato compatibile con il tool *Pipe3*, creati dall'*adapterA2* che ha bisogno in ingresso dei file nel formato Pnml_DISC;

4. rete1_p2.xml, rete2_p2.xml, rete3_p2.xml; tutti e tre i file sono restituiti in seguito all'esecuzione dell'*adapterA15_Pipe3toPipe2* che ha bisogno di avere in ingresso i file nel formato compatibile con il tool *Pipe3*, così da restituire i file, ovvero le reti, compatibili con il tool *Pipe2*;

5. reach_graph_1.txt, reach_graph_2.txt, reach_graph_3.txt; tutti e tre i file restituiti dal tool *Tina* rappresentano il grafo di raggiungibilità per ogni rete considerata, *Tina* riceve in ingresso soltanto reti compatibili con il tool *Pipe2*;

6. reach_graph_1.fsm, reach_graph_2.fsm, reach_graph_3.fsm; tutti e tre i file restituiti in seguito all'esecuzione del *pluginP7* rappresentano l'automa di cui si vuole studiare la diagnosticabilità, così da poter confrontare i risultati ottenuti con quelli che si hanno con i toolbox di Pocci e Perria.

Nello stesso script sono stati inseriti gli eseguibili *diag_UR* e *dcycle* così da avere il diagnosticatore e il numero di cicli indeterminati presenti nel sistema in considerazione, chiamati diagnoser_1.diag, diagnoser_2.diag, diagnoser_3.diag (e diagnoser_1.fsm, diagnoser_2.fsm, diagnoser_3.fsm) e inputFile_1.cycles, inputFile_2.cycles, inputFile_3.cycles. Per poter tenere traccia dei tempi impiegati per la simulazione, sono stati inseriti dei contatori, e si è fatto si che i valori numerici vengano memorizzati nel file risultati_22_Desuma.txt, in questo file vengono trascritti per ogni rete creata i seguenti tempi:

1. TIME_R, tempo espresso in secondi per il calcolo dell'automa di cui si vuole studiare la diagnosticabilità;

2. TIME_DIAG, tempo espresso in secondi per il calcolo del diagnosticatore;

3. TIME_CICLI, tempo espresso in secondi per il calcolo del numero di cicli indeterminati;

4. TIME_UMDES, tempo espresso in secondi per avere informazioni sulla diagnosticabilità del sistema, non è altro che la somma dei tempi sopra citati.

Nella Tabella 6.3 riportiamo i risultati ottenuti per le quattro reti create, grazie all'esecuzione degli script: script_desuma, riportato in Appendice A.3.1 e

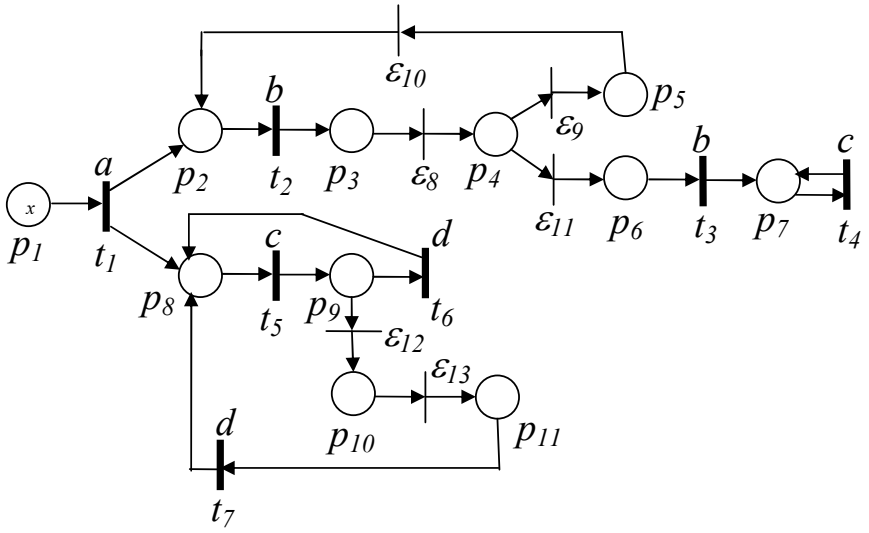

Figura 6.4: Esempio di rete di Petri con due classi di guasto in cui il numero di gettoni presenti nel posto p_1 varia da 1 a 3

script_tools, riportato in Appendice A.3.2. La sua struttura è uguale a quella della Tabella 6.2, con la differenza che ora la prima colonna indica il numero di gettoni presenti nel posto p_1:

Osservando la Tabella 6.3 vediamo come al crescere del numero dei gettoni la cardinalità dei grafi aumenta e di conseguenza aumenta anche il tempo (in secondi) per calcolarli. In questo esempio abbiamo fatto variare il numero di gettoni da uno a tre. Si può vedere che aumentando il numero dei gettoni fino a 3 con il tool *PN_DIAG*, nonostante la cardinalità di tutti i grafi calcolati ($|R|$, $|BRG|$, $|MBRG|$, $|BRD|$) sia cresciuta all'aumentare del numero dei gettoni, la verifica della diagnosticabilità non ha richiesto un tempo eccessivo. Il software *PN_DIAG_2* pur impiegando più tempo, è riuscito comunque a fornire dei risultati in tempi accettabbili. Osservando i risultati ottenuti con *UMDES* vediamo subito che per un numero di gettoni superiore a due l'eseguibile *diag_UR* non riesce a costruire il diagnosticatore in tempi ammissibili, è stato lasciato in simulazione per più di 24 ore senza aver generato nulla, per questo motivo nella tabella è stato scritto $o.t.$ (out of time), nel caso in cui il corrispondente programma non possa essere terminato entro il limite prefissato delle 24 ore. Stesso discorso vale per l'eseguibile *dcycle*, che può essere calcolato anche senza il diagnosticatore, perchè in ingresso prende anche esso l'automa, ma in ogni caso per poter dire qualcosa avremo dovuto verificare che gli eventuali cicli indeterminati trovati fossero corretti, e questo sarebbe stato possibile solo guardando il diagnosticatore che non è stato costruito. Di conseguenza nell'ultima colonna

UMDES								
RETE	*# nodi*	$t_R[s]$	*# stati diag*	$t_{diag}[s]$	*# cicli indet*	$t_{cicli}[s]$	$t_{UMDES}[s]$	*Diag?*
1	25	1.2513	20	0.0932	15	0.0565	0.0570	no
2	235	30.6945	125	13.3531	202	16.7693	16.7698	no
3	1355	247.5110	*o.t.*	*o.t.*	*o.t*	*o.t.*	*o.t.*	n.c.

PN_DIAG										
x	$\|R\|$	$t_R[s]$	$\|BRG\|$	$t_{BRG}[s]$	$\|MBRG\|$	$t_{MBRG}[s]$	$\|BRD\|$	$t_{BRD}[s]$	$t_{diag}[s]$	*Diag?*
1	25	0.0883	7	0.3735	13	0.1419	20	0.6135	1.0424	no
2	235	0.5773	25	0.6190	73	1.9592	142	2.3087	4.3703	no
3	1355	25.1544	65	9.6441	273	37.9437	750	47.4605	85.5183	no

PN_DIAG_2								
x	$\|MBRG\|$	$t_{MBRG}[s]$	$\|MBRD\|$	$t_{MBRD}[s]$	*classe guasto*	*# c. incerti*	$t_{diag}[s]$	*Diag?*
1	13	0.3417	20	0.5287	una due	6 11	1.1341	no
2	73	1.9119	142	54.2390	una due	30 83	56.7706	no
3	273	37.6123	750	6731.2	una due	92 331	6954.7	no

Tabella 6.3: Risultati della simulazione effettuata con i toolbox *UMDES*, *PN_DIAG*, *PN_DIAG_2* prendendo in considerazione la rete in Figura 6.4

della tabella in corrispondenza della presenza di tre gettoni nella marcatura iniziale si è scritto *n.c.* (not computable), in quanto senza il diagnosticatore e senza sapere il numero di cicli indeterminati non possiamo dedurre la diagnosticabilità del sistema.

A dimostrazione di quanto appena detto facciamo due esempi.

Esempio 6.2.1. *Consideriamo la rete in Figura 6.5 in cui l'insieme delle transizioni osservabili è* $T_o = \{t_1, t_2, t_3, t_4\}$*, l'insieme delle transizioni non osservabili è* $T_u = \{\varepsilon_5, \varepsilon_6\}$*. Sono presenti due classi di guasto* $T_f = T_f^1 \cup T_f^2 = \{\varepsilon_5\} \cup \{\varepsilon_6\}$*. La funzione di etichettatura è tale che:* $\mathscr{L}(t_1) = a$*,* $\mathscr{L}(t_2) = b$*,* $\mathscr{L}(t_3) = c$ *e* $\mathscr{L}(t_4) = d$*. Il numero di gettoni nel posto* p_1 *varia da* 1 *a* 6*.*

I risultati forniti dai toolbox PN_DIAG *e* PN_DIAG_2 *sono già stati riportati nel Capitolo 4 nelle Tabelle 4.3 e 4.4, Riportiamo nella Tabella 6.4 solo i risultati forniti da* UMDES*, la struttura è uguale a quella della prima parte della Tabella 6.2.*

Come si può vedere, essendo il numero dei nodi degli automi non eccessivamente grande gli eseguibili diag_UR *e* dcycle *riescono a portare dei risultati in tempi anche brevi.* ■

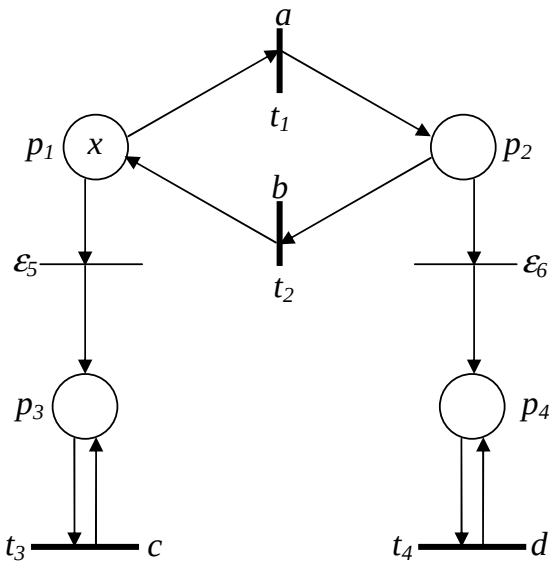

Figura 6.5: Esempio di rete di Petri con due classi di guasto in cui il numero di gettoni nel posto p_1 varia da 1 a 6

UMDES								
RETE	*# nodi*	$t_R[s]$	*# stati diag*	$t_{diag}[s]$	*# cicli indet*	$t_{cicli}[s]$	$t_{UMDES}[s]$	*Diag?*
1	4	1.4461	4	0.2949	0	0.0957	0.0962	si
2	10	1.1232	8	0.0968	3	0.0493	0.0497	no
3	20	1.1129	12	0.0413	17	0.0403	0.0407	no
4	35	1.1708	16	0.3101	80	0.0483	0.0488	no
5	56	1.1895	20	0.2962	300	0.0748	0.0752	no
6	84	1.2001	24	0.1582	1025	0.2016	0.2020	no

Tabella 6.4: Risultati della simulazione effettuata con *UMDES* prendendo in considerazione la rete in Figura 6.5

Esempio 6.2.2. *Consideriamo la rete in Figura 6.6 in cui l'insieme delle transizioni osservabili è* $T_o = \{t_1, t_2, t_3, t_4\}$*, l'insieme delle transizioni non osservabili è* $T_u = \{\varepsilon_5, \varepsilon_6, \varepsilon_7, \varepsilon_8\}$*, l'insieme delle transizioni di guasto è* $T_f = T_f^1 \cup T_f^2$*, dove* $T_f^1 = \{\varepsilon_7\}$ *e* $T_f^2 = \{\varepsilon_8\}$*, le etichette associate alla transizioni sono:* $\mathscr{L}(t_1) = a$*,* $\mathscr{L}(t_2) = \mathscr{L}(t_3) = b$ *e* $\mathscr{L}(t_4) = c$*. Il numero di gettoni nel posto* p_1 *varia da* 1 *a* 6*.*

Riportiamo nella Tabella 6.5 i risultati forniti da UMDES*, dal toolbox* PN_DIAG *e da* PN_DIAG_2*. La struttura è uguale a quella della Tabella 6.3.*

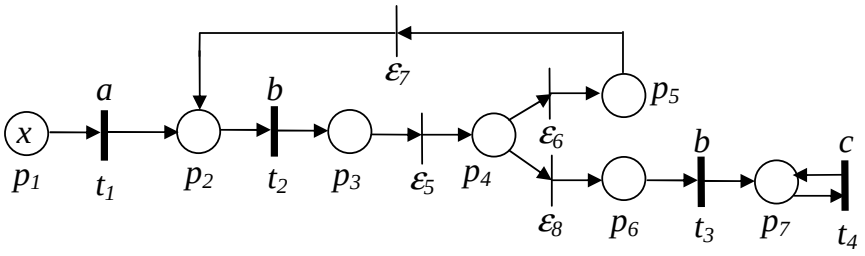

Figura 6.6: Esempio di rete di Petri con due classi di guasto, dove varia il numero di gettoni nel posto $p1$

Come si può vedere in questo caso risulta migliore il metodo usato dal toolbox PN_DIAG_2*, fornisce i risultati sulla diagnosticabilità in tempi brevi anche al crescere della cardinalità dello spazio di stato del grafo di raggiungibilità. Al contrario il toolbox* PN_DIAG *restituisce risultati circa la diagnosticabilità solo fino ad un numero di gettoni nel posto* p_1 *pari a* 3*, da* 4 *gettoni in poi, richiede un tempo, per determinare la diagnosticabilità non ammissibile, se posto a confronto con il tempo che impiega il software* PN_DIAG_2*. Per questo motivo nella Tabella 6.5, relativamente alla parte del tool* PN_DIAG *è stato scritto* $o.t.$ *(out of time) e* $n.c$ *(not computable), in quanto non ha fornito risultati entro* 3 *ore, e di conseguenza non è stato possibile stabilire se il sistema è diagnosticabile o meno (nel caso di* 4*,* 5 *e* 6 *gettoni nel posto* p_1*). Gli eseguibili* diag_UR *e* dcycle *riescono a calcolare il diagnosticatore e il numero di cicli indeterminati solo fino a* 5 *gettoni nel posto* p_1*, dopodichè nella Tabella 6.5 si è scritto* $o.t.$ *(out of time) e* $n.c$ *(not computable), in quanto non ha fornito risultati entro* 24 *ore, e di conseguenza non è stato possibile stabilire se il sistema è diagnosticabile o meno. Inoltre, come già detto nel Capitolo 5, nel caso di* 1 *gettone nel posto* p_1 *l'eseguibile* dcycle *fornisce un risultato errato, affermando che il sistema è non diagnosticabile, perchè trova un ciclo indeterminato non presente (vedere Figure 5.6 e 5.7).* ■

6.3 Modello d'esempio

Il modello fisico considerato descrive una famiglia di sistemi manifatturieri. Il sistema è composto da due gruppi di lavoro perfettamente simmetrici per operatività e finalità, atti, quindi, alla produzione dello stesso prodotto. Ciascun gruppo è caratterizzato da m linee di produzione, su cui vengono pilotate m differenti

UMDES								
RETE	*# nodi*	$t_R[s]$	*# stati diag*	$t_{diag}[s]$	*# cicli indet*	$t_{cicli}[s]$	$t_{UMDES}[s]$	*Diag?*
1	7	1.4701	7	0.0894	1	0.0715	0.0719	no
2	28	2.5942	22	0.1365	4	0.0614	0.0618	no
3	84	0.0597	50	0.0857	10	0.0700	0.0704	no
4	210	16.8971	95	1.1059	20	1.3284	1.3291	no
5	462	40.7375	161	157.8243	35	389.4166	389.4171	no
6	924	92.8037	*o.t.*	*o.t.*	*o.t.*	*o.t.*	*o.t.*	*n.c.*

PN_DIAG										
x	$\|R\|$	$t_R[s]$	$\|BRG\|$	$t_{BRG}[s]$	$\|MBRG\|$	$t_{MBRG}[s]$	$\|BRD\|$	$t_{BRD}[s]$	$t_{diag}[s]$	*Diag?*
1	7	0.0570	4	0.1569	5	0.0687	7	0.2673	0.5232	si
2	28	0.0112	10	0.0619	15	0.0963	25	0.2101	1.0168	no
3	84	0.0964	20	0.1964	35	0.2995	70	0.6319	12.1226	no
4	210	0.6386	35	0.6441	70	1.0198	167	2.4471	*o.t.*	*n.c.*
5	462	3.8944	56	2.2966	126	3.4223	364	9.4022	*o.t.*	*n.c.*
6	924	16.6454	84	6.9677	210	10.8613	750	33.8201	*o.t.*	*n.c.*

PN_DIAG_2								
x	$\|MBRG\|$	$t_{MBRG}[s]$	$\|MBRD\|$	$t_{MBRD}[s]$	*classe guasto*	*# c. incerti*	$t_{diag}[s]$	*Diag?*
1	5	0.3168	7	0.1300	una due	0 1	0.6308	si
2	15	0.1052	22	0.4721	una due	2 2	0.6604	no
3	35	0.3253	50	3.8884	una due	7 3	4.2734	no
4	70	1.0185	95	30.5814	una due	16 4	31.9234	no
5	126	3.3809	161	215.1705	una due	30 5	220.6367	no
6	210	11.3820	252	1412.2	una due	50 6	1434.8	no

Tabella 6.5: Risultati della simulazione effettuata con i toolbox *UMDES*, *PN_DIAG* e *PN_DIAG_2* prendendo in considerazione la rete in Figura 6.6

parti dello stesso componente. Ciascuna linea di produzione effettua l differenti operazioni, modellate con l transizioni regolari ε_i ($i = 1,\ldots,l$), prima di poter fornire ciascuno dei pezzi. Sono presenti $m-1$ transizioni di guasto, rappresentate da transizioni non osservabili f_i ($i = 1,\ldots,m-1$). Si noti che il guasto f_i comporta lo spostamento accidentale di un pezzo dalla linea di produzione i a quella $i+1$. La terminazione della i-esima catena di l operazioni è modellata con la transizione osservabile etichettata a_i. Questo comporta che, nonostante tutte le parti del componente risultino lavorate correttamente, ossia siano presenti all'appello $2m$ pezzi finiti, alcune di esse abbiano subito un trattamento anziché un altro, compromettendo la funzionalità del prodotto assemblato. Affinchè si possa verificare la diagnosticabilità del sistema, si è pensato di etichettare in

due differenti maniere le transizioni osservabili. Nello specifico, nel caso in cui la prima transizione del primo gruppo di lavoro produca la stessa osservazione della seconda, si verificherebbe che il guasto non sia diagnosticabile.

Il sistema è caratterizzato da tre parametri:

- $2m$, il numero totale di linee di produzione;
- l, il numero di operazioni che devono essere effettuate su ciascun componente del prodotto;
- d, una variabile binaria che comporta alcune caratteristiche nella funzione di etichettatura. In particolare per $d = 0$ il sistema non è diagnosticabile, mentre lo è per $d = 1$.

Il modello analizzato è mostrato in Figura 6.7. Le transizioni osservabili sono evidenziate in verde, le transizioni non osservabili ma regolari sono evidenziate in blu, mentre quelle non osservabili di guasto in rosso. L'ellisse verde mette in evidenza la transizione osservabile a_{2-d}, dipendente dal parametro d. Se $d = 0$ la transizione sarà a_2, mentre se $d = 1$ la transizione sarà a_1.

6.3.1 Risultati numerici

Poichè la correzione dei *plugin* e degli *adapter* ha richiesto più tempo del previsto, non è stato possibile effettuare l'intera simulazione del modello parametrico per i due toolbox *PN_DIAG* e *PN_DIAG_2*, ma ci limitiamo a riportare i risultati che si erano ottenuti nella tesi di laurea di Perria [26] nella Tabella 6.6. Lo script per effettuare questa simulazione sarebbe stato quello riportato in Appendice A.4.2. I risultati ottenuti con i toolbox *PN_DIAG* e *PN_DIAG_2* devono essere confrontati con quelli ottenuti utilizzando il metodo proposto da Lafortune [30]. Quindi è stato necessario scrivere lo script: script_desuma riportati in Appendice A.4.1, in cui si considerano tutte le reti di Petri prese in considerazione dai toolbox precedenti, per un totale di dodici reti da analizzare, ma soprattutto da convertire in automi (questo grazie ai convertitori presenti nella piattaforma). I risultati ottenuti in seguito all'esecuzione di script_desuma sono riportati nella prima parte della Tabella 6.6 che è strutturata nel seguente modo:

- le colonne 1, 2 e 3 indicano i valori assunti dai parametri m, l e d;
- la colonna 4 indica il numero di nodi dell'automa a stati finiti ($\#\ nodi$);

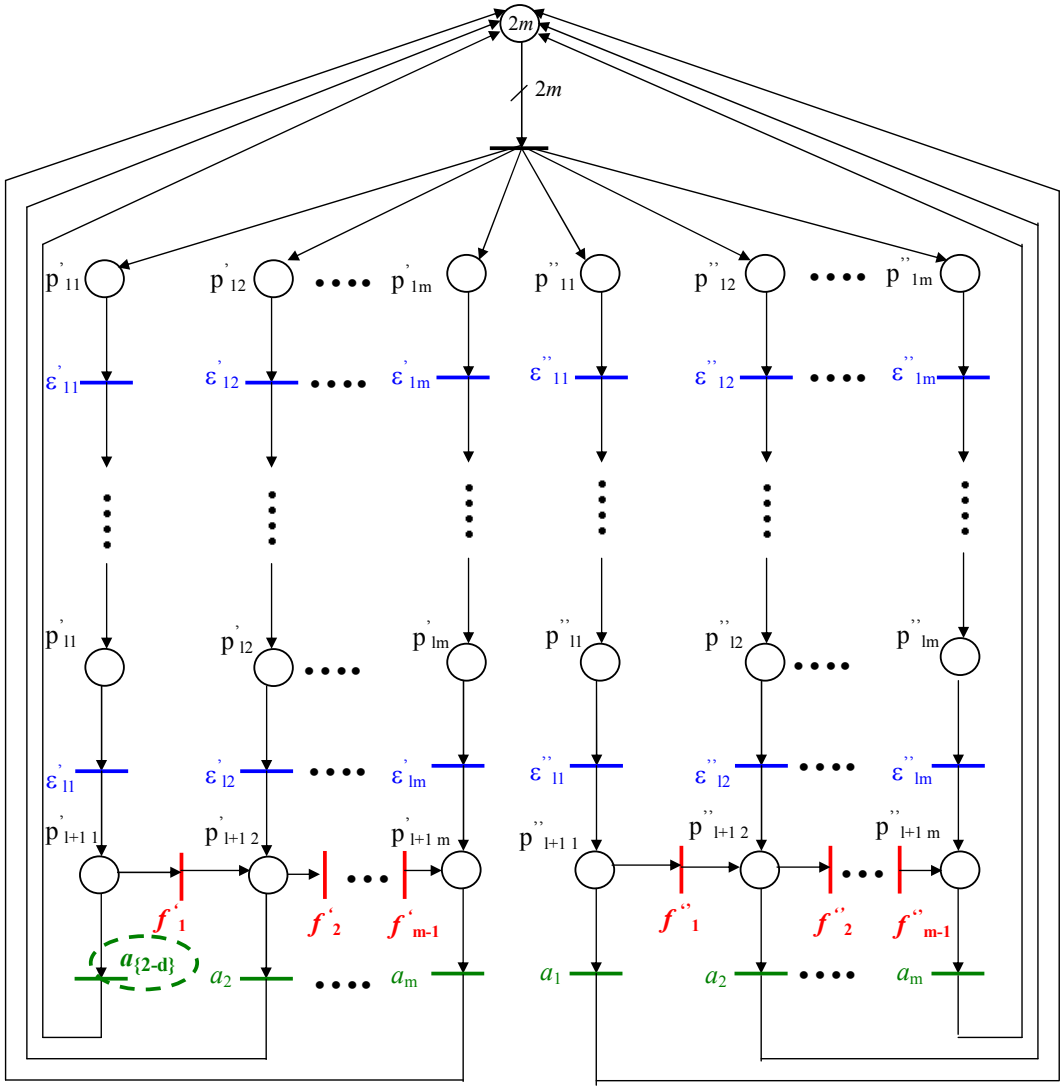

Figura 6.7: Modello parametrico

- la colonna 5 indica il tempo espresso in secondi per generare l'automa ($t_R[s]$);
- la colonna 6 indica il numero di stati del diagnosticatore (# $stati\ diag$);
- la colonna 7 indica il tempo per il calcolo del diagnosticatore espresso in secondi ($t_{diag}[s]$);
- la colonna 8 indica il numero di cicli indeterminati trovati tramite *dcycle* (# $cicli\ indet$);
- la colonna 9 indica il tempo di calcolo espresso in secondi per la ricerca di cicli indeterminati ($t_{cicli}[s]$);
- la colonna 10 indica il tempo totale espresso in secondi per la verifica della diagnosticabilità ($t_{UMDES}[s]$);

La seconda parte della Tabella 6.6 è strutturata in questo modo:

- le colonne 1, 2 e 3 indicano i valori assunti dai parametri m, l e d;
- la colonna 4 indica la cardinalità del grafo di raggiungibilità ($|R|$);
- le colonne 5 e 6 indicano la cardinalità del BRG ($|BRG|$) e il tempo espresso in secondi impiegato per calcolarlo ($t_{BRG}[s]$), rispettivamente;
- le colonne 7 e 8 indicano la cardinalità dell'MBRG ($|MBRG|$) e il tempo espresso in secondi impiegato per calcolarlo ($t_{MBRG}[s]$), rispettivamente;
- le colonne 9 e 10 indicano la cardinalità del BRD ($|BRD|$) e il tempo espresso in secondi impiegato per calcolarlo ($t_{BRD}[s]$), rispettivamente;
- la colonna 11 indica il tempo espresso in secondi impiegato per valutare la diagnosticabilità del sistema ($t_{diag}[s]$).

La terza parte della Tabella 6.6 è strutturata in questo modo:

- le colonne 1, 2 e 3 indicano i valori assunti dai parametri m, l e d;
- le colonne 4 e 5 indicano la cardinalità dell' MBRG ($|MBRG|$) e il tempo espresso in secondi impiegato per calcolarlo ($t_{MBRG}[s]$), rispettivamente;
- le colonne 6 e 7 indicano la cardinalità dell'MBRD ($|MBRD|$) e il tempo espresso in secondi impiegato per calcolarlo ($t_{MBRD}[s]$), rispettivamente;

Desuma									
m	l	d	$\#\ nodi$	$t_R[s]$	$\#\ stati\ diag$	$t_{diag}[s]$	$\#\ cicli\ indet$	$t_{cicli}[s]$	$t_{UMDES}[s]$
2	1	0	121	9.4380	23	1.1108	4	1.1047	1.1052
2	1	1	121	9.1268	18	1.1095	0	1.1029	1.1034
2	2	0	361	49.0930	23	59.4199	4	60.1832	60.1837
2	2	1	361	49.3635	18	30.0448	0	29.7378	29.7383
2	3	0	841	151.6197	*o.t.*	*o.t.*	*o.t.*	*o.t.*	*n.c.*
2	3	1	841	151.3285	*o.t.*	*o.t.*	*o.t.*	*o.t.*	*n.c.*
3	1	0	2025	492.2394	*o.t.*	*o.t.*	*o.t.*	*o.t.*	*n.c.*
3	1	1	2025	152.9325	*o.t.*	*o.t.*	*o.t.*	*o.t.*	*n.c.*
3	2	0	10000	223.9785	*o.t.*	*o.t.*	*o.t.*	*o.t.*	*n.c.*
3	2	1	10000	755.0693	*o.t.*	*o.t.*	*o.t.*	*o.t.*	*n.c.*
3	3	0	34225	7663.64	*o.t.*	*o.t.*	*o.t.*	*o.t.*	*n.c.*
3	3	1	34225	2473.3	*o.t.*	*o.t.*	*o.t.*	*o.t.*	*n.c.*

PN_DIAG										
m	l	d	$\|R\|$	$\|BRG\|$	$t_{BRG}[s]$	$\|MBRG\|$	$t_{MBRG}[s]$	$\|BRD\|$	$t_{BRD}[s]$	$t_{diag}[s]$
2	1	0	121	16	0.2507	36	0.2489	23	0.3742	0.81924
2	1	1	121	16	0.1241	36	0.2286	19	0.1693	0.41527
2	2	0	361	16	0.8591	36	1.3919	23	0.9416	2.506
2	2	1	361	16	0.8951	36	1.4162	19	0.9416	2.4549
2	3	0	841	16	6.1768	36	8.9129	23	6.2582	15.2573
2	3	1	841	16	6.0672	36	8.8495	19	6.1128	14.9776
3	1	0	2025	64	21.997	484	54.796	67	24.0085	83.9516
3	1	1	2025	64	22.0254	484	55.225	55	23.266	3269.69
3	2	0	10000	64	1073.82	484	1952.07	67	1075.82	3032.92
3	2	1	10000	64	1063.88	484	1948.61	55	1065.13	6094.96
3	3	0	34225	64	25023.46	484	42769.45	67	25025.47	67800
3	3	1	34225	64	24037.15	484	41650.06	55	24038.54	68756.6

PN_DIAG_2							
m	l	d	$\|MBRG\|$	$t_{MBRG}[s]$	$\|MBRD\|$	$t_{MBRD}[s]$	$t_{diag}[s]$
2	1	0	36	0.2732	23	0.6637	10.9585
2	1	1	36	0.2324	18	0.3175	0.559042
2	2	0	36	1.3947	23	0.6232	11.8428
2	2	1	36	1.3975	18	0.3223	1.7317
2	3	0	36	9.0693	23	0.6096	19.4819
2	3	1	36	8.989	18	0.3157	9.30709
3	1	0	484	58.097	67	1148.85	520932.2
3	1	1	484	56.473	54	672.148	728.673
3	2	0	484	2009.37	67	1162.29	523640.04
3	2	1	484	2011.049	54	682.25	2694.006
3	3	0	484	41141.35	67	1168.19	563037.6
3	3	1	484	41188.35	54	684.63	41873

Tabella 6.6: Risultati delle simulazioni effettuate con *UMDES*, il toolbox *PN_DIAG* e *PN_DIAG_2*

- la colonna 8 indica il tempo espresso in secondi impiegato per valutare la diagnosticabilità del sistema ($t_{diag}[s]$).

Considerando le ultime due parti della Tabella 6.6, ovvero quelle che si riferiscono ai software *PN_DIAG* e *PN_DIAG_2*, è possibile fare le seguenti osservazioni:

- la cardinalità del BRG varia solo in funzione del parametro m mentre quella del grafo di raggiungibilità (R) varia sia in funzione di m che in funzione di l, inoltre si può osservare come la cardinalità del BRG è sempre molto inferiore rispetto alla cardinalità di R;
- la cardinalità del BRG e dell'$MBRG$ è invariante rispetto ai parametri l e d, mentre aumenta al crescere del parametro m, ossia del numero di linee di produzione di ciascun gruppo di lavoro;
- la cardinalità del BRD e dell'$MBRD$ varia sia rispetto al parametro m sia rispetto al parametro d, ossia alla variabile che stabilisce se il sistema è diagnosticabile oppure no;
- per quanto riguarda i tempi di calcolo variano principalmente rispetto al numero di linee di produzione (m) e rispetto al numero di lavorazioni (l). Per $m = 2$ i tempi sono relativamente bassi, per $m = 3$ i tempi crescono notevolmente al crescere di l da 1 a 3.

Per quanto riguarda i tempi per il calcolo della diagnosticabilità, si verifica è che per $d = 0$ (ossia quando il sistema è non diagnosticabile) si ottengono dei risultati migliori con il software *PN_DIAG* mentre per $d = 1$ (ossia quando il sistema è diagnosticabile) si ottengono risultati migliori con il software *PN_DIAG_2*. Questo si verifica perché, nel caso in cui il sistema sia non diagnosticabile, con il software realizzato da Pocci [28] viene analizzato un ciclo per volta e quindi è sufficiente che venga individuato un ciclo indeterminato nel $MBRG$ per ogni classe di guasto per affermare che il sistema è non diagnosticabile. Nel software realizzato da Perria [26] si ha una maggior perdita di tempo nella costruzione del $MBRD$ (perchè bisogna considerare un numero maggiore di marcature) e nel calcolo dei cicli in quanto prima della verifica della diagnosticabilità per ogni classe di guasto, si ha la preventiva determinazione di tutti i cicli incerti nel $MBRD$ per quella classe di guasto.

Nel caso in cui il sistema sia diagnosticabile ($d = 1$) è necessario andare ad analizzare tutti i cicli presenti per poter affermare che lo sia, per cui il software *PN_DIAG* impiega un tempo maggiore in quanto è necessario andare a determinare anche i percorsi che conducono ai cicli.

Resta ora da confrontare i risultati avuti con *Umdes*. Si nota subito che se il numero di nodi dell'automa è un numero abbastanza grande, *diag_UR* non riesce a costruire il diagnosticatore di cui abbiamo bisogno per dedurre la diagnosticabilità del sistema. E' anche possibile che per sistemi di grandi dimensioni gli eseguibili *diag_UR* e *dcycle* presentino un baco che non gli permette di effettuare le operazioni per i quali sono stati progettati. Per questo motivo, abbiamo nuovamente contattato il Professor Lafortune, dell'università del Michigan, così da poter verificare l'eventuale presenza di bachi, che mandano in loop gli eseguibili posti sotto esame. Al momento l'approccio migliore per verificare la diagnosticabilità di un sistema, anche di grandi dimensioni, è quello fornito dal tool *PN_DIAG_2*.

Tutte le simulazioni sono state eseguite con un notebook, avente come sistema operativo Windows 7 a 64 bit, con processore Intel(R)Core(TM)i3 CPU con una frequenza di clock pari a 2.27GHz e 4GB di RAM.

Capitolo 7

Conclusioni

7.1 Conclusioni e sviluppi futuri

Inizialmente lo scopo della tesi era quello di utilizzare la *Piattaforma Software* per confrontare i diversi approcci di diagnosi, e dedurre quale sia al momento quello ottimale. Si è fatto riferimento agli approcci inerenti le reti di Petri, alla base dei software realizzati da Pocci [28] e Perria [26], e alla procedura proposta dal Professor Lafortune [30]. Per poter effettuare i confronti è stato necessario studiare la *Piattaforma Software* sviluppata da un consorzio di partner europei guidato dall'università di Cagliari, che permette l'integrazione di strumenti software per l'analisi e il controllo di *sistemi ad eventi discreti*. In particolare, la piattaforma considera come principali modelli di riferimento le reti di Petri e gli automi.

Per poterla utilizzare nel pieno delle sue funzionalità, è stato obbligatorio testarla nelle sue varie componenti. In particolare, sono stati fatti numerosi test sui convertitori, parte fondamentale della piattaforma, in quanto uno degli scopi per cui è stata creata è appunto quello di poter passare da un formato di file ad un altro. I test hanno evidenziato numerosi bachi nei *plugin* e negli *adapter*.

Per questo motivo, lo scopo della tesi è stato leggermente modificato, il primo obiettivo è diventato quello di capire i problemi presenti nei convertitori in modo da fornire tutto il supporto necessario per risolvere i bachi presenti, agli sviluppatori di tali convertitori e questo ha permesso l'eliminazione di tutti i bachi finora scoperti. Il processo di test, correzione dei convertitori e verifica è stato ripetuto varie volte con esempi random. Una volta che tutti i bachi sono stati

eliminati si è potuto procedere al confronto tra i diversi approcci di diagnosi.

Abbiamo testato la piattaforma prima con esempi abbastanza semplici per verificare che tutto funzionasse correttamente sia con i toolbox realizzati da Pocci e Perria, sia con quello riferito agli automi, ovvero *UMDES*, che determina la diagnosticabilità di un dato automa attraverso gli eseguibili *diag_UR* e *dcycle*. Poi si è passati ad utilizzare la piattaforma e di conseguenza i toolbox in essa contenuti con un modello parametrico nel formato rete di Petri, da esso siamo potuti passare al modello automa. Quindi, effettuato questo passaggio abbiamo potuto constatare la grande utilità della piattaforma.

Una volta che la *Piattaforma Software* è stata resa funzionale, grazie alla collaborazione di Akhela srl si è potuto affrontare il problema della diagnosticabilità dei *sistemi ad eventi discreti*, facendo riferimento agli automi e alle reti di Petri limitate.

Per quanto riguarda il software di Pocci [28] si basa sul concetto di *marcatura di base*, e la verifica della diagnosticabilità è basata sull'analisi di due grafi il *Modified Basis Reachability Graph* (MBRG) e il *Basis Reachability Diagnoser* (BRD), mentre nel software di Perria [26] la verifica della diagnosticabilità è basata sull'analisi di due grafi il *Modified Basis Reachability Graph* (MBRG) e il *Modified Basis Reachability Diagnoser* (MBRD).

Di entrambi si è studiata la diagnosticabilità, e questo ha messo in evidenza come i toolbox *PN_DIAG* e *PN_DIAG_2* forniscano dei risultati anche per sistemi di grandi dimensioni. Al contrario il tool *UMDES* non termina, nel caso in cui il numero di nodi dell'automa di cui vogliamo verificare la diagnosticabilità sia intorno ai 600, in questi casi: siamo stati costretti ad interrompere l'esecuzione degli eseguibili utili alla nostra verifica dopo aver aspettato più di 24 ore un risultato. Al momento l'approccio migliore per verificare la diagnosticabilità di un sistema, anche di grandi dimensioni, è quello fornito dal tool *PN_DIAG_2*, perchè il tool *PN_DIAG* all'aumentare dei parametri, richiede dei tempi eccessivi dovendo determinare tutti i percorsi che conducono ai cicli.

In futuro, sarebbe interessante effettuare un confronto in termini di complessità computazionale tra gli approcci relativi alle reti di Petri e quello agli automi considerando però una rete che non presenti transizioni non osservabili. Inoltre nell'immediato futuro la *Piattaforma Software* sarà resa ancora più funzionale, con l'aggiunta di nuovi toolbox, che richiederanno la scrittura di nuovi *adapter* e *plugin*. E di conseguenza saranno necessari nuovi test per verificare che tutto funzioni perfettamente entro la fine del progetto *DISC*.

Appendice A

File Matlab e Script Matlab

Sommario

In questo capitolo vengono inseriti i file Matlab che racchiudono le operazioni svolte dai toolbox *PN_DIAG* e *PN_DIAG_2*. Inoltre si riportano i listati degli script, tramite i quali la piattaforma permette di effettuare le conversioni da un formato ad un altro, così da poter effettuare l'analisi della diagnosticabilità delle varie Reti di Petri considerate durante il lavoro di tesi.

A.1 File Matlab

A.1.1 esempio1_Pocci.m

Riportiamo il file esempio1_Pocci.m utilizzato per verificare la diagnosticabilità della Rete di Petri in Figura 4.1 utilizzando il toolbox *PN_DIAG*.

```
% Esempio 1 Pocci

Pre = [0 0 0 1 1 0 0 0;
       1 0 0 0 0 0 0 0;
       0 1 0 0 0 1 0 0;
       0 0 0 0 0 0 1 0;
       0 0 0 0 0 0 0 1;
       0 0 1 0 0 0 0 0];
Post = [0 1 1 0 0 0 0 0;
        0 0 0 1 0 0 0 0;
        1 0 0 0 1 0 0 0;
        0 0 0 0 0 1 0 0;
        0 0 0 0 0 0 1 0;
```

```
        0 0 0 0 0 0 0 1];
M0=[2,0,0,0,0,0]';
F={[5,7]};
L={[1];[2,3]};
E={['a'];['b']};

% Esempio_DISC

fid = fopen('risultati_esempio1.txt','wt');
if (fid < 0)
    error('could not open file "risultati_esempio1.txt"');
end

% Creazione del grafo di raggiungibilità
tic
G = graph(Pre,Post,M0);
t_R = toc;
[sizeG,lenghtG] = size(G);
fprintf(fid,'|R| = %d\n',sizeG-1);
fprintf(fid,'TIME_R = %d\n',t_R);

% Creazione del Basis Reachability Graph (BRG)
tic
brg = BRG(Pre,Post,M0,F,L,E);
t_brg = toc;
fprintf(fid,'|brg| = %d\n',brg{end,1});
fprintf(fid,'TIME_brg = %d\n',t_brg);

% Creazione del Modified Basis Reachability Graph (MBRG)
tic
mbrg = MBRG(Pre,Post,M0,F,L,E);
t_mbrg = toc;
fprintf(fid,'|mbrg| = %d\n',mbrg{end,1});
fprintf(fid,'TIME_mbrg = %d\n',t_mbrg);

% Creazione del Basis Reachability Diagnoser (BRD)
tic
brd = BRD(brg,Pre,Post,M0,F,L,E);
t_brd = toc;
fprintf(fid,'|brd| = %d\n',brd{end,1});
fprintf(fid,'TIME_brd = %d\n',t_brd+t_brg);

% Verifica della diagnosticabilità
tic
diagnosability22(mbrg,brd)
t_diagnos = toc;
fprintf(fid,'TIME_diagnosability = %d\n', t_diagnos+t_brd+t_brg+t_mbrg);

fclose(fid);
```

A.1.2 esempio1_Perria.m

Riportiamo il file esempio1_Perria.m utilizzato per verificare la diagnosticabilità della Rete di Petri in Figura 4.1 utilizzando il toolbox *PN_DIAG_2*.

```
% Esempio 1 Perria
Pre = [0 0 0 1 1 0 0 0;
```

```
       1 0 0 0 0 0 0 0;
       0 1 0 0 0 1 0 0;
       0 0 0 0 0 0 1 0;
       0 0 0 0 0 0 0 1;
       0 0 1 0 0 0 0 0];
Post = [0 1 1 0 0 0 0 0;
        0 0 0 1 0 0 0 0;
        1 0 0 0 1 0 0 0;
        0 0 0 0 0 1 0 0;
        0 0 0 0 0 0 1 0;
        0 0 0 0 0 0 0 1];
M0=[2,0,0,0,0,0]'; F={[5,7]}; L={[1];[2,3]}; E={['a'];['b']};

% Esempio DISC
fid = fopen('risultati_esempio1.txt','wt'); if (fid < 0)
    error('could not open file "risultati_esempio1.txt"');
end

% Creazione del Modified Basis Reachability Graph (MBRG)
tic
T = MBRG(Pre,Post,M0,F,L,E);
t_mbrg = toc;
fprintf(fid,'|mbrg| = %d\n',T{end,1});
fprintf(fid,'TIME_mbrg = %d\n',t_mbrg);

% Creazione del Modified Basis Reachability Diagnoser (MBRD)
tic
[D,numTclas,sizeE,no,nf] = MBRD(T,Pre,Post,M0,F,L,E);
t_mbrd = toc;
fprintf(fid,'|mbrd| = %d\n',D{end,1});
fprintf(fid,'TIME_mbrd = %d\n',t_mbrd);

% Verifica della diagnosticabilità del sistema
tic
MBRD_diag_ott(D,T,sizeE,no,nf)
t_diagnos = toc;
fprintf(fid,'TIME_diagnosability = %d\n',t_diagnos+t_mbrg+t_mbrd)

fclose(fid);
```

A.1.3 esempio2_Pocci

Riportiamo il file esempio2_Pocci.m utilizzato per verificare la diagnosticabilità della Rete di Petri in Figura 4.2 utilizzando il toolbox *PN_DIAG*.

```
% Esempio 2 Pocci
fid = fopen('risultati_esempio2.txt','wt');
if (fid < 0)
    error('could not open file "risultati_esempio2.txt"');
end

for x = 1:6
Pre = [1 0 0 0 1 0;
       0 1 0 0 0 1;
       0 0 1 0 0 0;
       0 0 0 1 0 0];
Post = [0 1 0 0 0 0;
```

```
        1 0 0 0 0 0;
        0 0 1 0 1 0;
        0 0 0 1 0 1];
M0 = [x 0 0 0]';
F = {[5];[6]};
L = {[1];[2];[3];[4]};
E = {['a'];['b'];['c'];['d']};
fprintf(fid,'\n x=%d \n',x);

%Esempio_DISC

% Creazione del grafo di raggiungibilità
tic
G = graph(Pre,Post,M0);
t_R = toc;
[sizeG,lenghtG] = size(G);
fprintf(fid,'|R| = %d\n',sizeG-1);
fprintf(fid,'TIME_R = %d\n',t_R);

% Creazione del Basis Reachability Graph (BRG)
tic
brg = BRG(Pre,Post,M0,F,L,E);
t_brg = toc;
fprintf(fid,'|brg| = %d\n',brg{end,1});
fprintf(fid,'TIME_brg = %d\n',t_brg);

% Creazione del Modified Basis Reachability Graph (MBRG)
tic
mbrg = MBRG(Pre,Post,M0,F,L,E);
t_mbrg = toc;
fprintf(fid,'|mbrg| = %d\n',mbrg{end,1});
fprintf(fid,'TIME_mbrg = %d\n',t_mbrg);

% Creazione del Basis Reachability Diagnoser (BRD)
tic
brd = BRD(brg,Pre,Post,M0,F,L,E);
t_brd = toc;
fprintf(fid,'|brd| = %d\n',brd{end,1});
fprintf(fid,'TIME_brd = %d\n',t_brd+t_brg);

% Verifica della diagnosticabilità
tic
diagnosability22(mbrg,brd)
t_diagnos = toc;
fprintf(fid,'TIME_diagnosability = %d\n',t_diagnos+t_brd+t_brg+t_mbrg);
end

fclose(fid);
```

A.1.4 esempio2_Perria

Riportiamo il file esempio2_Perria.m utilizzato per verificare la diagnosticabilità della Rete di Petri in Figura 4.2 utilizzando il toolbox *PN_DIAG_2*.

```
% Esempio 2 Perria
fid = fopen('risultati_esempio2.txt','wt');
if (fid < 0)
```

```
    error('could not open file "risultati_esempio2.txt"');
end

for x = 1:6
Pre = [1 0 0 0 1 0;
       0 1 0 0 0 1;
       0 0 1 0 0 0;
       0 0 0 1 0 0];
Post = [0 1 0 0 0 0;
        1 0 0 0 0 0;
        0 0 1 0 1 0;
        0 0 0 1 0 1];
M0 = [x 0 0 0]';
F = {[5];[6]};
L = {[1];[2];[3];[4]};
E = {['a'];['b'];['c'];['d']};
fprintf(fid,'\n x=%d \n',x);

% Esempio DISC

% Creazione del Modified Basis Reachability Graph (MBRG)
tic
T = MBRG(Pre,Post,M0,F,L,E);
t_mbrg = toc;
fprintf(fid,'|mbrg| = %d\n',T{end,1});
fprintf(fid,'TIME_mbrg = %d\n',t_mbrg);

% Creazione del Modified Basis Reachability Diagnoser (MBRD)
tic
[D,numTclas,sizeE,no,nf] = MBRD(T,Pre,Post,M0,F,L,E);
t_mbrd = toc;
fprintf(fid,'|mbrd| = %d\n',D{end,1});
fprintf(fid,'TIME_mbrd = %d\n',t_mbrd);

% Verifica della diagnosticabilità del sistema
tic
MBRD_diag_ott(D,T,sizeE,no,nf)
t_diagnos = toc;
fprintf(fid,'TIME_diagnosability = %d\n',t_diagnos+t_mbrg+t_mbrd);
end
fclose(fid);
```

A.1.5 main_PN_DIAG.m

Esempio_DISC. Attenzione tale file è da richiamare dentro uno script della piattaforma solo se in esso è stata inserita la funzione fopen che apre un file di testo, che contiene la cardinalità e i tempi dei grafi calcolati, e restituisce una variabile pn_diag che individua il file all'interno del programma.

```
% Creazione del grafo di raggiungibilità
tic
G = graph(Pre,Post,M0);
t_R = toc;
[sizeG,lenghtG] = size(G);
fprintf(pn_diag,'|R| = %d\n',sizeG-1);
fprintf(pn_diag,'TIME_R = %d\n',t_R);
```

```
% Creazione del Basis Reachability Graph (BRG)
tic
brg = BRG(Pre,Post,M0,F,L,E);
t_brg = toc;
fprintf(pn_diag,'|brg| = %d\n',brg{end,1});
fprintf(pn_diag,'TIME_brg = %d\n',t_brg);

% Creazione del Modified Basis Reachability Graph (MBRG)
tic
mbrg = MBRG(Pre,Post,M0,F,L,E);
t_mbrg = toc;
fprintf(pn_diag,'|mbrg| = %d\n',mbrg{end,1});
fprintf(pn_diag,'TIME_mbrg = %d\n',t_mbrg);

% Creazione del Basis Reachability Diagnoser (BRD)
tic
brd = BRD(brg,Pre,Post,M0,F,L,E);
t_brd = toc;
fprintf(pn_diag,'|brd| = %d\n',brd{end,1});
fprintf(pn_diag,'TIME_brd = %d\n',t_brd+t_brg);

% Verifica della diagnosticabilità
tic
diagnosability22(mbrg,brd)
t_diagnos = toc;
fprintf(pn_diag,'TIME_diagnosability = %d\n',t_diagnos+t_brd+t_brg+t_mbrg);
```

A.1.6 launch_DISC.m

Esempio_DISC. Attenzione tale file è da richiamare dentro uno script della piattaforma solo se in esso è stata inserita la funzione fopen che apre un file di testo, che contiene la cardinalità e i tempi dei grafi calcolati, e restituisce una variabile pn_diag_2 che individua il file all'interno del programma.

```
% Creazione del Modified Basis Reachability Graph (MBRG)
tic
T = MBRG(Pre,Post,M0,F,L,E);
t_mbrg = toc;
fprintf(pn\_diag\_2,'|mbrg| = %d\n',T{end,1});
fprintf(pn\_diag\_2,'TIME_mbrg = %d\n',t_mbrg);

% Creazione del Modified Basis Reachability Diagnoser (MBRD)
tic
[D,numTclas,sizeE,no,nf] = MBRD(T,Pre,Post,M0,F,L,E);
t_mbrd = toc;
fprintf(pn\_diag\_2,'|mbrd| = %d\n',D{end,1});
fprintf(pn\_diag\_2,'TIME_mbrd = %d\n',t_mbrd);

% Verifica della diagnosticabilità del sistema
tic
MBRD_diag_ott(D,T,sizeE,no,nf)
t_diagnos = toc;
fprintf(pn\_diag\_2,'TIME_diagnosability = %d\n',t_diagnos+t_mbrg+t_mbrd);
```

A.1.7 simulation.m

Questo file matlab serve unicamente per richiamare il modello parametrico. Viene richiamato solo all'interno dello script_tools per poter effettuare la verifica della diagnosticabilità del modello parametrico con i toolbox "PN_DIAG" e "PN_DIAG_2"

```
[Pre, Post, M0, L, E, F] = model(2*m,l,d,1);
```

A.2 Script seguendo il primo modo

Riportiamo gli script utilizzati per lo studio della rete di Petri in Figura 6.2 utilizzata per spiegare il primo modo di utilizzo della *Piattaforma Software*.

A.2.1 script_desuma.m

Questo script serve per verificare la diagnosticabilità delle reti di Petri facendo uso degli eseguibili contenuti nella libreria UMDES di Desuma. Tutti i file restituiti dagli *adapter* e dai *plugin* utilizzati vengono salvati nella cartella Output_Files. Mentre i tempi calcolati all'interno dello script vengono salvati in un apposito file di testo. In particolare in questo script viene analizzata la diagnosticabilità di sei reti di Petri.

```
clear all
desuma = fopen('C:\Users\Laura\Desktop\Tesi\rete3\
               Output_Files\Desuma_3.txt','wt');
if (desuma < 0)
    error('could not open file "C:\Users\Laura\Desktop\
    Tesi\rete3\Output_Files\Desuma_3.txt"');
end

fprintf(desuma, '\n********** RETE 3_1 **********\n');
tic
!adapterA15_Pipe3toPipe2.exe "rete3_1_p3.xml"
"Output_Files\rete3_1_p2.xml"
tic
!tina -R -PNML "Output_Files\rete3_1_p2.xml" "Output_Files\reach_graph_1.txt"
!pluginP7.exe "Output_Files\reach_graph_1.txt" "Output_Files\reach_graph_1.fsm"
t_R_1 = toc;
fprintf(desuma,'TIME_R = %d\n',t_R_1);
tic
!diag_UR.exe "Output_Files\reach_graph_1.fsm" failure_1.ft
"Output_Files\diagnoser_1.diag" "Output_Files\diagnoser_1.fsm" 2
t_diag_1 = toc;
fprintf(desuma,'TIME_DIAG = %d\n',t_diag_1);
tic
!dcycle.exe "Output_Files\reach_graph_1.fsm" failure_1.ft
```

```
"Output_Files\inputFile_1.cycles"
t_cicli_1 = toc;
fprintf(desuma,'TIME_CICLI = %d\n',t_cicli_1);
t_UMDES_1 = toc;
fprintf(desuma,'TIME_UMDES = %d\n',t_UMDES_1);
!pluginP1.exe "rete3_1_p3.xml" "Output_Files\rete3_1_pnml.pnml"
!adapterA1.exe "Output_Files\rete3_1_pnml.pnml"
"Output_Files\rete3_1_matlab_disc.m"

fprintf(desuma, '\n********** RETE 3_2 **********\n');
tic
!adapterA15_Pipe3toPipe2.exe "rete3_2_p3.xml"
"Output_Files\rete3_2_p2.xml"
tic
!tina -R -PNML "Output_Files\rete3_2_p2.xml" "Output_Files\reach_graph_2.txt"
!pluginP7.exe "Output_Files\reach_graph_2.txt" "Output_Files\reach_graph_2.fsm"
t_R_2 = toc;
fprintf(desuma,'TIME_R = %d\n',t_R_2);
tic
!diag_UR.exe "Output_Files\reach_graph_2.fsm" failure_2.ft
"Output_Files\diagnoser_2.diag" "Output_Files\diagnoser_2.fsm" 2
t_diag_2 = toc;
fprintf(desuma,'TIME_DIAG = %d\n',t_diag_2);
tic
!dcycle.exe "Output_Files\reach_graph_2.fsm" failure_2.ft
"Output_Files\inputFile_2.cycles"
t_cicli_2 = toc;
fprintf(desuma,'TIME_CICLI = %d\n',t_cicli_2);
t_UMDES_2 = toc;
fprintf(desuma,'TIME_UMDES = %d\n',t_UMDES_2);
!pluginP1.exe "rete3_2_p3.xml" "Output_Files\rete3_2_pnml.pnml"
!adapterA1.exe "Output_Files\rete3_2_pnml.pnml"
"Output_Files\rete3_2_matlab_disc.m"

fprintf(desuma, '\n********** RETE 3_3 **********\n');
tic
!adapterA15_Pipe3toPipe2.exe "rete3_3_p3.xml"
"Output_Files\rete3_3_p2.xml"
tic
!tina -R -PNML "Output_Files\rete3_3_p2.xml" "Output_Files\reach_graph_3.txt"
!pluginP7.exe "Output_Files\reach_graph_3.txt" "Output_Files\reach_graph_3.fsm"
t_R_3 = toc;
fprintf(desuma,'TIME_R = %d\n',t_R_3);
tic
!diag_UR.exe "Output_Files\reach_graph_3.fsm" failure_3.ft
"Output_Files\diagnoser_3.diag" "Output_Files\diagnoser_3.fsm" 2
t_diag_3 = toc;
fprintf(desuma,'TIME_DIAG = %d\n',t_diag_3);
tic
!dcycle.exe "Output_Files\reach_graph_3.fsm" failure_3.ft
"Output_Files\inputFile_3.cycles"
t_cicli_3 = toc;
fprintf(desuma,'TIME_CICLI = %d\n',t_cicli_3);
t_UMDES_3 = toc;
fprintf(desuma,'TIME_UMDES = %d\n',t_UMDES_3);
!pluginP1.exe "rete3_3_p3.xml" "Output_Files\rete3_3_pnml.pnml"
!adapterA1.exe "Output_Files\rete3_3_pnml.pnml"
"Output_Files\rete3_3_matlab_disc.m"

fprintf(desuma, '\n********** RETE 3_4 **********\n');
tic
!adapterA15_Pipe3toPipe2.exe "rete3_4_p3.xml"
```

```
"Output_Files\rete3_4_p2.xml"
tic
!tina -R -PNML "Output_Files\rete3_4_p2.xml" "Output_Files\reach_graph_4.txt"
!pluginP7.exe "Output_Files\reach_graph_4.txt" "Output_Files\reach_graph_4.fsm"
t_R_4 = toc;
fprintf(desuma,'TIME_R = %d\n',t_R_4);
tic
!diag_UR.exe "Output_Files\reach_graph_4.fsm" failure_4.ft
"Output_Files\diagnoser_4.diag" "Output_Files\diagnoser_4.fsm" 2
t_diag_4 = toc;
fprintf(desuma,'TIME_DIAG = %d\n',t_diag_4);
tic
!dcycle.exe "Output_Files\reach_graph_4.fsm" failure_4.ft
"Output_Files\inputFile_4.cycles"
t_cicli_4 = toc;
fprintf(desuma,'TIME_CICLI = %d\n',t_cicli_4);
t_UMDES_4 = toc;
fprintf(desuma,'TIME_UMDES = %d\n',t_UMDES_4);
!pluginP1.exe "rete3_4_p3.xml" "Output_Files\rete3_4_pnml.pnml"
!adapterA1.exe "Output_Files\rete3_4_pnml.pnml"
"Output_Files\rete3_4_matlab_disc.m"

fprintf(desuma, '\n********** RETE 3_5 **********\n');
tic
!adapterA15_Pipe3toPipe2.exe "rete3_5_p3.xml"
"Output_Files\rete3_5_p2.xml"
tic
!tina -R -PNML "Output_Files\rete3_5_p2.xml" "Output_Files\reach_graph_5.txt"
!pluginP7.exe "Output_Files\reach_graph_5.txt" "Output_Files\reach_graph_5.fsm"
t_R_5 = toc;
fprintf(desuma,'TIME_R = %d\n',t_R_5);
tic
!diag_UR.exe "Output_Files\reach_graph_5.fsm" failure_5.ft
"Output_Files\diagnoser_5.diag" "Output_Files\diagnoser_5.fsm" 2
t_diag_5 = toc;
fprintf(desuma,'TIME_DIAG = %d\n',t_diag_5);
tic
!dcycle.exe "Output_Files\reach_graph_5.fsm" failure_5.ft
"Output_Files\inputFile_5.cycles"
t_cicli_5 = toc;
fprintf(desuma,'TIME_CICLI = %d\n',t_cicli_5);
t_UMDES_5 = toc;
fprintf(desuma,'TIME_UMDES = %d\n',t_UMDES_5);
!pluginP1.exe "rete3_5_p3.xml" "Output_Files\rete3_5_pnml.pnml"
!adapterA1.exe "Output_Files\rete3_5_pnml.pnml"
"Output_Files\rete3_5_matlab_disc.m"

fprintf(desuma, '\n********** RETE 3_6 **********\n');
tic
!adapterA15_Pipe3toPipe2.exe "rete3_6_p3.xml"
"Output_Files\rete3_6_p2.xml"
tic
!tina -R -PNML "Output_Files\rete3_6_p2.xml" "Output_Files\reach_graph_6.txt"
!pluginP7.exe "Output_Files\reach_graph_6.txt" "Output_Files\reach_graph_6.fsm"
t_R_6 = toc;
fprintf(desuma,'TIME_R = %d\n',t_R_6);
tic
!diag_UR.exe "Output_Files\reach_graph_6.fsm" failure_6.ft
"Output_Files\diagnoser_6.diag" "Output_Files\diagnoser_6.fsm" 2
t_diag_6 = toc;
fprintf(desuma,'TIME_DIAG = %d\n',t_diag_6);
tic
```

```
!dcycle.exe "Output_Files\reach_graph_6.fsm" failure_6.ft
"Output_Files\inputFile_6.cycles"
t_cicli_6 = toc;
fprintf(desuma,'TIME_CICLI = %d\n',t_cicli_6);
t_UMDES_6 = toc;
fprintf(desuma,'TIME_UMDES = %d\n',t_UMDES_6);
!pluginP1.exe "rete3_6_p3.xml" "Output_Files\rete3_6_pnml.pnml"
!adapterA1.exe "Output_Files\rete3_6_pnml.pnml"
"Output_Files\rete3_6_matlab_disc.m"

fclose(desuma);
```

A.2.2 script_tools

Questo script serve per verificare la diagnosticabilità delle reti di Petri facendo uso dei toolbox *PN_DIAG* e *PN_DIAG_2*. I tempi impiegati per la costruzione dei grafi contenuti nei due toolbox e la cardinalità degli stessi, vengono memorizzati in un file di testo contenuto nella cartella Output_Files. In questo script vengono richiamati i file main_PN_DIAG.m (per *PN_DIAG*) e launch_DISC.m (per *PN_DIAG_2*).

```
clear all
pn_diag = fopen('C:\Users\Laura\Desktop\Tesi\rete3\
                Output_Files\PN_DIAG_3.txt','wt');
if (pn_diag < 0)
   error('could not open file "C:\Users\Laura\Desktop\Tesi\rete3\
         Output_Files\PN_DIAG_3.txt"');
end
pn_diag_2 = fopen('C:\Users\Laura\Desktop\Tesi\rete3\
                    Output_Files\PN_DIAG_2_3.txt','wt');
if (pn_diag_2 < 0)
   error('could not open file "C:\Users\Laura\Desktop\Tesi\rete3\
         Output_Files\PN_DIAG_2_3.txt"');
end

fprintf('\n********** RETE 3_1 **********\n');
!adapterA9.exe "Output_Files\rete3_1_matlab_disc.m"
"Output_Files\rete3_1_input.m"
run('C:\Users\Laura\Desktop\Tesi\rete3\Output_Files\rete3_1_input.m')
fprintf('\n********** TOOLBOX PN_DIAG **********\n');
fprintf(pn_diag, '\n********** RETE 3_1 **********\n');
run('C:\Akhela\WP4SoftwarePlatform\Tools\PN_DIAG\main_PN_DIAG.m')
fprintf('\n********** TOOLBOX PN_DIAG_2 **********\n');
fprintf(pn_diag_2, '\n********** RETE 3_1 **********\n');
run('C:\Akhela\WP4SoftwarePlatform\Tools\PN_DIAG_2\launch_DISC.m')

fprintf('\n********** RETE 3_2 **********\n');
!adapterA9.exe "Output_Files\rete3_2_matlab_disc.m"
"Output_Files\rete3_2_input.m"
run('C:\Users\Laura\Desktop\Tesi\rete3\Output_Files\rete3_2_input.m')
fprintf('\n********** TOOLBOX PN_DIAG **********\n');
fprintf(pn_diag, '\n********** RETE 3_2 **********\n');
run('C:\Akhela\WP4SoftwarePlatform\Tools\PN_DIAG\main_PN_DIAG.m')
fprintf('\n********** TOOLBOX PN_DIAG_2 **********\n');
```

```
fprintf(pn_diag_2, '\n********** RETE 3_2 **********\n');
run('C:\Akhela\WP4SoftwarePlatform\Tools\PN_DIAG_2\launch_DISC.m')

fprintf('\n********** RETE 3_3 **********\n');
!adapterA9.exe "Output_Files\rete3_3_matlab_disc.m"
"Output_Files\rete3_3_input.m"
run('C:\Users\Laura\Desktop\Tesi\rete3\Output_Files\rete3_3_input.m')
fprintf('\n********** TOOLBOX PN_DIAG **********\n');
fprintf(pn_diag, '\n********** RETE 3_3 **********\n');
run('C:\Akhela\WP4SoftwarePlatform\Tools\PN_DIAG\main_PN_DIAG.m')
fprintf('\n********** TOOLBOX PN_DIAG_2 **********\n');
fprintf(pn_diag_2, '\n********** RETE 3_3 **********\n');
run('C:\Akhela\WP4SoftwarePlatform\Tools\PN_DIAG_2\launch_DISC.m')

fprintf('\n********** RETE 3_4 **********\n');
!adapterA9.exe "Output_Files\rete3_4_matlab_disc.m"
"Output_Files\rete3_4_input.m"
run('C:\Users\Laura\Desktop\Tesi\rete3\Output_Files\rete3_4_input.m')
fprintf('\n********** TOOLBOX PN_DIAG **********\n');
fprintf(pn_diag, '\n********** RETE 3_4 **********\n');
run('C:\Akhela\WP4SoftwarePlatform\Tools\PN_DIAG\main_PN_DIAG.m')
fprintf('\n********** TOOLBOX PN_DIAG_2 **********\n');
fprintf(pn_diag_2, '\n********** RETE 3_4 **********\n');
run('C:\Akhela\WP4SoftwarePlatform\Tools\PN_DIAG_2\launch_DISC.m')

fprintf('\n********** RETE 3_5 **********\n');
!adapterA9.exe "Output_Files\rete3_5_matlab_disc.m"
"Output_Files\rete3_5_input.m"
run('C:\Users\Laura\Desktop\Tesi\rete3\Output_Files\rete3_5_input.m')
fprintf('\n********** TOOLBOX PN_DIAG **********\n');
fprintf(pn_diag, '\n********** RETE 3_5 **********\n');
run('C:\Akhela\WP4SoftwarePlatform\Tools\PN_DIAG\main_PN_DIAG.m')
fprintf('\n********** TOOLBOX PN_DIAG_2 **********\n');
fprintf(pn_diag_2, '\n********** RETE 3_5 **********\n');
run('C:\Akhela\WP4SoftwarePlatform\Tools\PN_DIAG_2\launch_DISC.m')

fprintf('\n********** RETE 3_6 **********\n');
!adapterA9.exe "Output_Files\rete3_6_matlab_disc.m"
"Output_Files\rete3_6_input.m"
run('C:\Users\Laura\Desktop\Tesi\rete3\Output_Files\rete3_6_input.m')
fprintf('\n********** TOOLBOX PN_DIAG **********\n');
fprintf(pn_diag, '\n********** RETE 3_6 **********\n');
run('C:\Akhela\WP4SoftwarePlatform\Tools\PN_DIAG\main_PN_DIAG.m')
fprintf('\n********** TOOLBOX PN_DIAG_2 **********\n');
fprintf(pn_diag_2, '\n********** RETE 3_6 **********\n');
run('C:\Akhela\WP4SoftwarePlatform\Tools\PN_DIAG_2\launch_DISC.m')

fclose(pn_diag);
fclose(pn_diag_2);
```

A.3 Script seguendo il secondo modo

Riportiamo gli script utilizzati per lo studio della rete di Petri in Figura 6.4 utilizzata per spiegare il secondo modo di utilizzo della piattaforma.

A.3.1 script_desuma.m

Questo script è stato scritto per poter risalire all'automa, corrispondente alla rete di Petri data, così da poterne studiare la diagnosticabilità. Nella prima parte dello script è presente il codice Matlab per far si che venga creato un numero di file pari al numero di variazioni del parametro, in questo caso, dovranno essere generati tre file. Nella seconda parte invece vi sono i comandi per lo studio vero e proprio della diagnosticabilità. I file generati in seguito alle numerose conversioni vengono salvati nella cartella Output_Files, e sempre in esse viene creato un file di testo che memorizza i tempi impiegati a svolgere l'analisi.

```
clear all
desuma = fopen('C:\Users\Laura\Desktop\Tesi\reti\rete22\
               Output_Files\risultati_22_desuma.txt','wt');
if (desuma < 0)
   error('could not open file "C:\Users\Laura\Desktop\Tesi\reti\rete22\
         Output_Files\risultati_22_desuma.txt"');
end

% Prima Parte

for x = 1:3
   path = strcat('rete', strcat(int2str(x), '.m'));
   rete22_input;
   fid = fopen(path, 'w');
   fprintf(fid, '%% Matlab_PN_DIAG input file\n');
   fprintf(fid, '%%\n');
   fprintf(fid, '%%Pre incidence matrices\n');
   fprintf(fid, '%%\n');
   [sizePre, lengthPre] = size(Pre);
   [sizePost, lengthPost] = size(Post);
   [sizeM0, lengthM0] = size(M0);
   [sizeL, lengthL] = size(L);
   [sizeE, lengthE] = size(E);
   [sizeF, lengthF] = size(F);
% Costruzione della Matrice di Pre-Incidenza
   fprintf(fid, 'Pre=[ ');
   for j = 1:sizePre
       for jj = 1:lengthPre
          fprintf(fid, '%d ', Pre(j,jj));
       end
          if(j ~= sizePre)
              fprintf(fid, ';\n');
          end
   end
   fprintf(fid, '];\n');
   fprintf(fid, '%%\n');
   fprintf(fid, '%%Post incidence matrices\n');
   fprintf(fid, '%%\n');
% Costruzione della Matrice di Post-Incidenza
   fprintf(fid, 'Post=[ ');
   for j = 1:sizePost
       for jj = 1:lengthPost
           fprintf(fid, '%d ', Post(j,jj));
       end
          if(j ~= sizePost)
```

```
            fprintf(fid, ';\n');
        end
    end
    fprintf(fid, '];\n');
    fprintf(fid, '%%\n');
    fprintf(fid, '%%Initial marking \n');
    fprintf(fid, '%%\n');
% Costruzione della marcatura iniziale
    fprintf(fid, 'M0=[ ');
    for j = 1:sizeM0
        for jj = 1:lengthM0
            fprintf(fid, '%d', M0(j,jj));
        end
        if(j ~= sizeM0)
            fprintf(fid, ' ');
        end
    end
    fprintf(fid, ' ]'';\n');
    fprintf(fid, '%%\n');
    fprintf(fid, '%%Matrix of cells of fault\n');
    fprintf(fid, '%%\n');
% Costruzione della Matrice di celle F
% (che definisce le classi di guasto)
    fprintf(fid, 'F={');
    for j = 1:sizeF
        [sizeFj, lengthFj] = size(F{j,1});
        fprintf(fid, '[');
        for jj = 1:lengthFj
            fprintf(fid, '%d', F{j}(1,jj));
            if(jj ~= lengthFj)
                fprintf(fid, ',');
            end
        end
        fprintf(fid, ']');
        if(j ~= sizeF)
            fprintf(fid, ';');
        end
    end
    fprintf(fid, '};\n');
    fprintf(fid, '%%\n');
    fprintf(fid, '%%Matrix of cells of the index of the transitions with label\n');
    fprintf(fid, '%%\n');
% Costruzione della Matrice di celle L
% (che definisce la funzione di etichettatura)
    fprintf(fid, '')
    fprintf(fid, 'L={');
    for j = 1:sizeL
        [sizeLj, lengthLj] = size(L{j,1});
        fprintf(fid, '[');
        for jj = 1:lengthLj
            fprintf(fid, '%d', L{j}(1,jj));
            if(jj ~= lengthLj)
                fprintf(fid, ',');
            end
        end
        fprintf(fid, ']');
        if(j ~= sizeL)
            fprintf(fid, ';');
        end
    end
    fprintf(fid, '};\n');
    fprintf(fid, '%%\n');
```

```
   fprintf(fid, '%%Matrix of cells of the label\n');
   fprintf(fid, '%%\n');
% Costruzione della Matrice di celle E
% (che definisce l'alfabeto)
   fprintf(fid, 'E={');
   for j = 1:sizeE
       fprintf(fid, '[');
       fprintf(fid, '''%s''', E{j}(1,1));
       fprintf(fid, ']');
       if(j ~= sizeE)
           fprintf(fid, ';');
       end
   end
   fprintf(fid, '};\n');
   fprintf(fid, '%%\n');
   fprintf(fid, '%% The array "net_array" describes this net\n');
   fprintf(fid, 'net_array={Pre,Post,M0,F,L,E};');
   fclose(fid);
end

% Seconda parte

fprintf(desuma, '\n***** RETE 1 *****\n');
run('C:\Users\Laura\Desktop\Tesi\reti\rete22\rete1.m')
fprintf(desuma, '\n x = 1 \n');
!pluginP9.exe "rete1.m" "Output_Files\rete1_matlab_disc.m"
!pluginP2.exe "Output_Files\rete1_matlab_disc.m" "Output_Files\rete1_pnml.pnml"
!adapterA2.exe "Output_Files\rete1_pnml.pnml" "Output_Files\rete1_p3.xml"
tic
!adapterA15_Pipe3toPipe2.exe "Output_Files\rete1_p3.xml"
"Output_Files\rete1_p2.xml"
tic
!tina -R -PNML "Output_Files\rete1_p2.xml" "Output_Files\reach_graph_1.txt"
!pluginP7.exe "Output_Files\reach_graph_1.txt" "Output_Files\reach_graph_1.fsm"
t_R_1 = toc;
fprintf(desuma,'TIME_R = %d\n',t_R_1);
tic
!diag_UR.exe "Output_Files\reach_graph.fsm" failure.ft
"Output_Files\diagnoser_1.diag" "Output_Files\diagnoser_1.fsm" 2
t_diag_1 = toc;
fprintf(desuma,'TIME_DIAG = %d\n',t_diag_1);
tic
!dcycle.exe "Output_Files\reach_graph_1.fsm" failure.ft
"Output_Files\inputFile_1.cycles"
t_cicli_1 = toc;
fprintf(desuma,'TIME_CICLI = %d\n',t_cicli_1);
t_UMDES_1 = toc;
fprintf(desuma,'TIME_UMDES = %d\n',t_UMDES_1);

fprintf(desuma, '\n***** RETE 2 *****\n');
run('C:\Users\Laura\Desktop\Tesi\reti\rete22\rete2.m')
fprintf(desuma, '\n x = 2 \n');
!pluginP9.exe "rete2.m" "Output_Files\rete2_matlab_disc.m"
!pluginP2.exe "Output_Files\rete2_matlab_disc.m" "Output_Files\rete2_pnml.pnml"
!adapterA2.exe "Output_Files\rete2_pnml.pnml" "Output_Files\rete2_p3.xml"
tic
!adapterA15_Pipe3toPipe2.exe "Output_Files\rete2_p3.xml"
"Output_Files\rete2_p2.xml"
tic
!tina -R -PNML "Output_Files\rete2_p2.xml" "Output_Files\reach_graph_2.txt"
!pluginP7.exe "Output_Files\reach_graph_2.txt" "Output_Files\reach_graph_2.fsm"
t_R_2 = toc;
```

```
fprintf(desuma,'TIME_R = %d\n',t_R_2);
tic
!diag_UR.exe "Output_Files\reach_graph_2.fsm" failure.ft
"Output_Files\diagnoser_2.diag" "Output_Files\diagnoser_2.fsm" 2
t_diag_2 = toc;
fprintf(desuma,'TIME_DIAG = %d\n',t_diag_2);
tic
!dcycle.exe "Output_Files\reach_graph_2.fsm" failure.ft
"Output_Files\inputFile_2.cycles"
t_cicli_2 = toc;
fprintf(desuma,'TIME_CICLI = %d\n',t_cicli_2);
t_UMDES_2 = toc;
fprintf(desuma,'TIME_UMDES = %d\n',t_UMDES_2);

fprintf(desuma, '\n***** RETE 3 *****\n');
run('C:\Users\Laura\Desktop\Tesi\reti\rete22\rete3.m')
fprintf(desuma, '\n x = 3 \n');
!pluginP9.exe "rete3.m" "Output_Files\rete3_matlab_disc.m"
!pluginP2.exe "Output_Files\rete3_matlab_disc.m" "Output_Files\rete3_pnml.pnml"
!adapterA2.exe "Output_Files\rete3_pnml.pnml" "Output_Files\rete3_p3.xml"
tic
!adapterA15_Pipe3toPipe2.exe "Output_Files\rete3_p3.xml"
"Output_Files\rete3_p2.xml"
tic
!tina -R -PNML "Output_Files\rete3_p2.xml" "Output_Files\reach_graph_3.txt"
!pluginP7.exe "Output_Files\reach_graph_3.txt" "Output_Files\reach_graph_3.fsm"
t_R_3 = toc;
fprintf(desuma,'TIME_R = %d\n',t_R_3);
tic
!diag_UR.exe "Output_Files\reach_graph_3.fsm" failure.ft
"Output_Files\diagnoser_3.diag" "Output_Files\diagnoser_3.fsm" 2
t_diag_3 = toc;
fprintf(desuma,'TIME_DIAG = %d\n',t_diag_3);
tic
!dcycle.exe "Output_Files\reach_graph_3.fsm" failure.ft
"Output_Files\inputFile_3.cycles"
t_cicli_3 = toc;
fprintf(desuma,'TIME_CICLI = %d\n',t_cicli_3);
t_UMDES_3 = toc;
fprintf(desuma,'TIME_UMDES = %d\n',t_UMDES_3);

fclose(desuma);
```

A.3.2 script_tools.m

Questo script è stato scritto per la verifica della diagnosticabilità della Rete di Petri in Figura 6.4 utilizzando i toolbox *PN_DIAG* e *PN_DIAG_2*.

```
clear all
pn_diag = fopen('C:\Users\Laura\Desktop\Tesi\reti\rete22\
                Output_Files\risultati_22_PN_DIAG.txt','wt');
if (pn_diag < 0)
   error('could not open file "C:\Users\Laura\Desktop\Tesi\reti\rete22\
          Output_Files\risultati_22_PN_DIAG.txt"');
end
pn_diag_2 = fopen('C:\Users\Laura\Desktop\Tesi\reti\rete22\
```

```
                    Output_Files\risultati_22_PN_DIAG_2.txt','wt');
if (pn_diag_2 < 0)
   error('could not open file "C:\Users\Laura\Desktop\Tesi\reti\rete22\
          Output_Files\risultati_22_PN_DIAG_2.txt"');
end

for x = 1:3

run('C:\Users\Laura\Desktop\Tesi\reti\rete22\rete22_input.m')
fprintf('\n********** TOOLBOX PN_DIAG **********\n');
fprintf(pn_diag, '\n x = %d \n', x);
run('C:\Akhela\WP4SoftwarePlatform\Tools\PN_DIAG\main_PN_DIAG.m')
fprintf('\n********** TOOLBOX PN_DIAG_2 **********\n');
fprintf(pn_diag_2, '\n x = %d \n', x);
run('C:\Akhela\WP4SoftwarePlatform\Tools\PN_DIAG_2\launch_DISC.m')
end

fclose(pn_diag);
fclose(pn_diag_2);
```

A.4 Script per il modello parametrico

Riportiamo gli script utilizzati per lo studio del modello parametrico in Figura 6.7.

A.4.1 script_desuma.m

```
clear all
desuma = fopen('C:\Users\Laura\Desktop\Tesi\reti\modello\
                Output_Files\model_desuma.txt','wt');
if (desuma < 0)
    error('could not open file "C:\Users\Laura\Desktop\Tesi\reti\modello\
           Output_Files\model_desuma.txt"');
end
i = 1;
   for m = 2:3
       for l = 1:3
           for d = 0:1
   path = strcat('rete', strcat(int2str(i), '.m'));
   [Pre, Post, M0, L, E, F] = model(2*m,l,d,1);
   fid = fopen(path, 'w');
   fprintf(fid, '%% Matlab_PN_DIAG input file\n');
   fprintf(fid, '%%\n');
   fprintf(fid, '%%Pre incidence matrices\n');
   fprintf(fid, '%%\n');
   [sizePre, lengthPre] = size(Pre);
   [sizePost, lengthPost] = size(Post);
   [sizeM0, lengthM0] = size(M0);
   [sizeL, lengthL] = size(L);
   [sizeE, lengthE] = size(E);
   [sizeF, lengthF] = size(F);
% Costruzione della Matrice di Pre-Incidenza
   fprintf(fid, 'Pre=[ ');
   for j = 1:sizePre
       for jj = 1:lengthPre
          fprintf(fid, '%d ', Pre(j,jj));
```

```
       end
          if(j ~= sizePre)
              fprintf(fid, ';\n');
          end
   end
   fprintf(fid, '];\n');
   fprintf(fid, '%%\n');
   fprintf(fid, '%%Post incidence matrices\n');
   fprintf(fid, '%%\n');
% Costruzione della Matrice di Post-Incidenza
   fprintf(fid, 'Post=[ ');
   for j = 1:sizePost
       for jj = 1:lengthPost
           fprintf(fid, '%d ', Post(j,jj));
       end
          if(j ~= sizePost)
              fprintf(fid, ';\n');
          end
   end
   fprintf(fid, '];\n');
   fprintf(fid, '%%\n');
   fprintf(fid, '%%Initial marking \n');
   fprintf(fid, '%%\n');
% Costruzione della marcatura iniziale
   fprintf(fid, 'M0=[ ');
   for j = 1:sizeM0
       for jj = 1:lengthM0
           fprintf(fid, '%d', M0(j,jj));
       end
          if(j ~= sizeM0)
              fprintf(fid, ' ');
          end
   end
   fprintf(fid, ' ]'';\n');
   fprintf(fid, '%%\n');
   fprintf(fid, '%%Matrix of cells of fault\n');
   fprintf(fid, '%%\n');
% Costruzione della Matrice di celle F
% (che definisce le classi di guasto)
   fprintf(fid, 'F={');
   for j = 1:sizeF
       [sizeFj, lengthFj] = size(F{j,1});
       fprintf(fid, '[');
       for jj = 1:lengthFj
           fprintf(fid, '%d', F{j}(1,jj));
           if(jj ~= lengthFj)
               fprintf(fid, ',');
           end
       end
       fprintf(fid, ']');
       if(j ~= sizeF)
           fprintf(fid, ';');
       end
   end
   fprintf(fid, '};\n');
   fprintf(fid, '%%\n');
   fprintf(fid, '%%Matrix of cells of the index of the transitions with label\n');
   fprintf(fid, '%%\n');
% Costruzione della Matrice di celle L
% (che definisce la funzione di etichettatura)
   fprintf(fid, '')
   fprintf(fid, 'L={');
```

```
   for j = 1:sizeL
       [sizeLj, lengthLj] = size(L{j,1});
       fprintf(fid, '[');
       for jj = 1:lengthLj
           fprintf(fid, '%d', L{j}(1,jj));
           if(jj ~= lengthLj)
               fprintf(fid, ',');
           end
       end
       fprintf(fid, ']');
       if(j ~= sizeL)
           fprintf(fid, ';');
       end
   end
   fprintf(fid, '};\n');
   fprintf(fid, '%%\n');
   fprintf(fid, '%%Matrix of cells of the label\n');
   fprintf(fid, '%%\n');
% Costruzione della Matrice di celle E
% (che definisce l'alfabeto)
   fprintf(fid, 'E={');
   for j = 1:sizeE
       fprintf(fid, '[');
       fprintf(fid, '''%s''', E{j}(1,1));
       fprintf(fid, ']');
       if(j ~= sizeE)
           fprintf(fid, ';');
       end
   end
   fprintf(fid, '};\n');
   fprintf(fid, '%%\n');
   fprintf(fid, '%% The array "net_array" describes this net\n');
   fprintf(fid, 'net_array={Pre,Post,M0,F,L,E};');
   fclose(fid);
   i = i + 1;
           end
       end
   end

fprintf(desuma, '\n***** RETE 1 *****\n');
run('C:\Users\Laura\Desktop\Tesi\reti\modello\rete1.m')
fprintf(desuma, '\n m = 2, l = 1, d = 0 \n');
!pluginP9.exe "rete1.m" "Output_Files\rete1_matlab_disc.m"
!pluginP2.exe "Output_Files\rete1_matlab_disc.m" "Output_Files\rete1_pnml.pnml"
!adapterA2.exe "Output_Files\rete1_pnml.pnml" "Output_Files\rete1_p3.xml"
tic
!adapterA15_Pipe3toPipe2.exe "Output_Files\rete1_p3.xml"
"Output_Files\rete1_p2.xml"
tic
!tina -R -PNML "Output_Files\rete1_p2.xml" "Output_Files\reach_graph_1.txt"
!pluginP7.exe "Output_Files\reach_graph_1.txt" "Output_Files\reach_graph_1.fsm"
t_R_1 = toc;
fprintf(desuma,'TIME_R = %d\n',t_R_1);
tic
!diag_UR.exe "Output_Files\reach_graph_1.fsm" failure_1.ft
"Output_Files\diagnoser_1.diag" "Output_Files\diagnoser_1.fsm" 2
t_diag_1 = toc;
fprintf(desuma,'TIME_DIAG = %d\n',t_diag_1);
tic
!dcycle.exe "Output_Files\reach_graph_1.fsm" failure_1.ft
"Output_Files\inputFile_1.cycles"
t_cicli_1 = toc;
```

```
fprintf(desuma,'TIME_CICLI = %d\n',t_cicli_1);
t_UMDES_1 = toc;
fprintf(desuma,'TIME_UMDES = %d\n',t_UMDES_1);

fprintf(desuma, '\n***** RETE 2 *****\n');
run('C:\Users\Laura\Desktop\Tesi\reti\modello\rete2.m')
fprintf(desuma, '\n m = 2, l = 1, d = 1 \n');
!pluginP9.exe "rete2.m" "Output_Files\rete2_matlab_disc.m"
!pluginP2.exe "Output_Files\rete2_matlab_disc.m" "Output_Files\rete2_pnml.pnml"
!adapterA2.exe "Output_Files\rete2_pnml.pnml" "Output_Files\rete2_p3.xml"
tic
!adapterA15_Pipe3toPipe2.exe "Output_Files\rete2_p3.xml"
"Output_Files\rete2_p2.xml"
tic
!tina -R -PNML "Output_Files\rete2_p2.xml" "Output_Files\reach_graph_2.txt"
!pluginP7.exe "Output_Files\reach_graph_2.txt" "Output_Files\reach_graph_2.fsm"
t_R_2 = toc;
fprintf(desuma,'TIME_R = %d\n',t_R_2);
tic
!diag_UR.exe "Output_Files\reach_graph_2.fsm" failure_2.ft
"Output_Files\diagnoser_2.diag" "Output_Files\diagnoser_2.fsm" 2
t_diag_2 = toc;
fprintf(desuma,'TIME_DIAG = %d\n',t_diag_2);
tic
!dcycle.exe "Output_Files\reach_graph_2.fsm" failure_2.ft
"Output_Files\inputFile_2.cycles"
t_cicli_2 = toc;
fprintf(desuma,'TIME_CICLI = %d\n',t_cicli_2);
t_UMDES_2 = toc;
fprintf(desuma,'TIME_UMDES = %d\n',t_UMDES_2);

fprintf(desuma, '\n***** RETE 3 *****\n');
run('C:\Users\Laura\Desktop\Tesi\reti\modello\rete3.m')
fprintf(desuma, '\n m = 2, l = 2, d = 0 \n');
!pluginP9.exe "rete3.m" "Output_Files\rete3_matlab_disc.m"
!pluginP2.exe "Output_Files\rete3_matlab_disc.m" "Output_Files\rete3_pnml.pnml"
!adapterA2.exe "Output_Files\rete3_pnml.pnml" "Output_Files\rete3_p3.xml"
tic
!adapterA15_Pipe3toPipe2.exe "Output_Files\rete3_p3.xml"
"Output_Files\rete3_p2.xml"
tic
!tina -R -PNML "Output_Files\rete3_p2.xml" "Output_Files\reach_graph_3.txt"
!pluginP7.exe "Output_Files\reach_graph_3.txt" "Output_Files\reach_graph_3.fsm"
t_R_3 = toc;
fprintf(desuma,'TIME_R = %d\n',t_R_3);
tic
!diag_UR.exe "Output_Files\reach_graph_3.fsm" failure_3.ft
"Output_Files\diagnoser_3.diag" "Output_Files\diagnoser_3.fsm" 2
t_diag_3 = toc;
fprintf(desuma,'TIME_DIAG = %d\n',t_diag_3);
tic
!dcycle.exe "Output_Files\reach_graph_3.fsm" failure_3.ft
"Output_Files\inputFile_3.cycles"
t_cicli_3 = toc;
fprintf(desuma,'TIME_CICLI = %d\n',t_cicli_3);
t_UMDES_3 = toc;
fprintf(desuma,'TIME_UMDES = %d\n',t_UMDES_3);

fprintf(desuma, '\n***** RETE 4 *****\n');
run('C:\Users\Laura\Desktop\Tesi\reti\modello\rete4.m')
fprintf(desuma, '\n m = 2, l = 2, d = 1 \n');
!pluginP9.exe "rete4.m" "Output_Files\rete4_matlab_disc.m"
```

```
!pluginP2.exe "Output_Files\rete4_matlab_disc.m" "Output_Files\rete4_pnml.pnml"
!adapterA2.exe "Output_Files\rete4_pnml.pnml" "Output_Files\rete4_p3.xml"
tic
!adapterA15_Pipe3toPipe2.exe "Output_Files\rete4_p3.xml"
"Output_Files\rete4_p2.xml"
tic
!tina -R -PNML "Output_Files\rete4_p2.xml" "Output_Files\reach_graph_4.txt"
!pluginP7.exe "Output_Files\reach_graph_4.txt" "Output_Files\reach_graph_4.fsm"
t_R_4 = toc;
fprintf(desuma,'TIME_R = %d\n',t_R_4);
tic
!diag_UR.exe "Output_Files\reach_graph_4.fsm" failure_4.ft
"Output_Files\diagnoser_4.diag" "Output_Files\diagnoser_4.fsm" 2
t_diag_4 = toc;
fprintf(desuma,'TIME_DIAG = %d\n',t_diag_4);
tic
!dcycle.exe "Output_Files\reach_graph_4.fsm" failure_4.ft
"Output_Files\inputFile_4.cycles"
t_cicli_4 = toc;
fprintf(desuma,'TIME_CICLI = %d\n',t_cicli_4);
t_UMDES_4 = toc;
fprintf(desuma,'TIME_UMDES = %d\n',t_UMDES_4);

fprintf(desuma, '\n***** RETE 5 *****\n');
run('C:\Users\Laura\Desktop\Tesi\reti\modello\rete5.m')
fprintf(desuma, '\n m = 2, l = 3, d = 0 \n');
!pluginP9.exe "rete5.m" "Output_Files\rete5_matlab_disc.m"
!pluginP2.exe "Output_Files\rete5_matlab_disc.m" "Output_Files\rete5_pnml.pnml"
!adapterA2.exe "Output_Files\rete5_pnml.pnml" "Output_Files\rete5_p3.xml"
tic
!adapterA15_Pipe3toPipe2.exe "Output_Files\rete5_p3.xml"
"Output_Files\rete5_p2.xml"
tic
!tina -R -PNML "Output_Files\rete5_p2.xml" "Output_Files\reach_graph_5.txt"
!pluginP7.exe "Output_Files\reach_graph_5.txt" "Output_Files\reach_graph_5.fsm"
t_R_5 = toc;
fprintf(desuma,'TIME_R = %d\n',t_R_5);
tic
!diag_UR.exe "Output_Files\reach_graph_5.fsm" failure_5.ft
"Output_Files\diagnoser_5.diag" "Output_Files\diagnoser_5.fsm" 2
t_diag_5 = toc;
fprintf(desuma,'TIME_DIAG = %d\n',t_diag_5);
tic
!dcycle.exe "Output_Files\reach_graph_5.fsm" failure_5.ft
"Output_Files\inputFile_5.cycles"
t_cicli_5 = toc;
fprintf(desuma,'TIME_CICLI = %d\n',t_cicli_5);
t_UMDES_5 = toc;
fprintf(desuma,'TIME_UMDES = %d\n',t_UMDES_5);

fprintf(desuma, '\n***** RETE 6 *****\n');
run('C:\Users\Laura\Desktop\Tesi\reti\modello\rete6.m')
fprintf(desuma, '\n m = 2, l = 3, d = 1 \n');
!pluginP9.exe "rete6.m" "Output_Files\rete6_matlab_disc.m"
!pluginP2.exe "Output_Files\rete6_matlab_disc.m" "Output_Files\rete6_pnml.pnml"
!adapterA2.exe "Output_Files\rete6_pnml.pnml" "Output_Files\rete6_p3.xml"
tic
!adapterA15_Pipe3toPipe2.exe "Output_Files\rete6_p3.xml"
"Output_Files\rete6_p2.xml"
tic
!tina -R -PNML "Output_Files\rete6_p2.xml" "Output_Files\reach_graph_6.txt"
!pluginP7.exe "Output_Files\reach_graph_6.txt" "Output_Files\reach_graph_6.fsm"
```

```
t_R_6 = toc;
fprintf(desuma,'TIME_R = %d\n',t_R_6);
tic
!diag_UR.exe "Output_Files\reach_graph_6.fsm" failure_6.ft
"Output_Files\diagnoser_6.diag" "Output_Files\diagnoser_6.fsm" 2
t_diag_6 = toc;
fprintf(desuma,'TIME_DIAG = %d\n',t_diag_6);
tic
!dcycle.exe "Output_Files\reach_graph_6.fsm" failure_6.ft
"Output_Files\inputFile_6.cycles"
t_cicli_6 = toc;
fprintf(desuma,'TIME_CICLI = %d\n',t_cicli_6);
t_UMDES_6 = toc;
fprintf(desuma,'TIME_UMDES = %d\n',t_UMDES_6);

fprintf(desuma, '\n***** RETE 7 *****\n');
run('C:\Users\Laura\Desktop\Tesi\reti\modello\rete7.m')
fprintf(desuma, '\n m = 3, l = 1, d = 0 \n');
!pluginP9.exe "rete7.m" "Output_Files\rete7_matlab_disc.m"
!pluginP2.exe "Output_Files\rete7_matlab_disc.m" "Output_Files\rete7_pnml.pnml"
!adapterA2.exe "Output_Files\rete7_pnml.pnml" "Output_Files\rete7_p3.xml"
tic
!adapterA15_Pipe3toPipe2.exe "Output_Files\rete7_p3.xml"
"Output_Files\rete7_p2.xml"
tic
!tina -R -PNML "Output_Files\rete7_p2.xml" "Output_Files\reach_graph_7.txt"
!pluginP7.exe "Output_Files\reach_graph_7.txt" "Output_Files\reach_graph_7.fsm"
t_R_7 = toc;
fprintf(desuma,'TIME_R = %d\n',t_R_7);
tic
!diag_UR.exe "Output_Files\reach_graph_7.fsm" failure_7.ft
"Output_Files\diagnoser_7.diag" "Output_Files\diagnoser_7.fsm" 2
t_diag_7 = toc;
fprintf(desuma,'TIME_DIAG = %d\n',t_diag_7);
tic
!dcycle.exe "Output_Files\reach_graph_7.fsm" failure_7.ft
"Output_Files\inputFile_7.cycles"
t_cicli_7 = toc;
fprintf(desuma,'TIME_CICLI = %d\n',t_cicli_7);
t_UMDES_7 = toc;
fprintf(desuma,'TIME_UMDES = %d\n',t_UMDES_7);

fprintf(desuma, '\n***** RETE 8 *****\n');
run('C:\Users\Laura\Desktop\Tesi\reti\modello\rete8.m')
fprintf(desuma, '\n m = 3, l = 1, d = 1 \n');
!pluginP9.exe "rete8.m" "Output_Files\rete8_matlab_disc.m"
!pluginP2.exe "Output_Files\rete8_matlab_disc.m" "Output_Files\rete8_pnml.pnml"
!adapterA2.exe "Output_Files\rete8_pnml.pnml" "Output_Files\rete8_p3.xml"
tic
!adapterA15_Pipe3toPipe2.exe "Output_Files\rete8_p3.xml"
"Output_Files\rete8_p2.xml"
tic
!tina -R -PNML "Output_Files\rete8_p2.xml" "Output_Files\reach_graph_8.txt"
!pluginP7.exe "Output_Files\reach_graph_8.txt" "Output_Files\reach_graph_8.fsm"
t_R_8 = toc;
fprintf(desuma,'TIME_R = %d\n',t_R_8);
tic
!diag_UR.exe "Output_Files\reach_graph_8.fsm" failure_8.ft
"Output_Files\diagnoser_8.diag" "Output_Files\diagnoser_8.fsm" 2
t_diag_8 = toc;
fprintf(desuma,'TIME_DIAG = %d\n',t_diag_8);
tic
```

```
!dcycle.exe "Output_Files\reach_graph_8.fsm" failure_8.ft
"Output_Files\inputFile_8.cycles"
t_cicli_8 = toc;
fprintf(desuma,'TIME_CICLI = %d\n',t_cicli_8);
t_UMDES_8 = toc;
fprintf(desuma,'TIME_UMDES = %d\n',t_UMDES_8);

fprintf(desuma, '\n***** RETE 9 *****\n');
run('C:\Users\Laura\Desktop\Tesi\reti\modello\rete9.m')
fprintf(desuma, '\n m = 3, l = 2, d = 0 \n'); !pluginP9.exe
"rete9.m" "Output_Files\rete9_matlab_disc.m" !pluginP2.exe
"Output_Files\rete9_matlab_disc.m" "Output_Files\rete9_pnml.pnml"
!adapterA2.exe "Output_Files\rete9_pnml.pnml"
"Output_Files\rete9_p3.xml" tic !adapterA15_Pipe3toPipe2.exe
"Output_Files\rete9_p3.xml" "Output_Files\rete9_p2.xml" tic !tina -R
-PNML "Output_Files\rete9_p2.xml" "Output_Files\reach_graph_9.txt"
!pluginP7.exe "Output_Files\reach_graph_9.txt"
"Output_Files\reach_graph_9.fsm" t_R_9 = toc;
fprintf(desuma,'TIME_R = %d\n',t_R_9);
tic
!diag_UR.exe "Output_Files\reach_graph_9.fsm" failure_9.ft
"Output_Files\diagnoser_9.diag" "Output_Files\diagnoser_9.fsm" 2
t_diag_9 = toc;
fprintf(desuma,'TIME_DIAG = %d\n',t_diag_9);
tic
!dcycle.exe "Output_Files\reach_graph_9.fsm" failure_9.ft
"Output_Files\inputFile_9.cycles"
t_cicli_9 = toc;
fprintf(desuma,'TIME_CICLI = %d\n',t_cicli_9);
t_UMDES_9 = toc;
fprintf(desuma,'TIME_UMDES = %d\n',t_UMDES_9);

fprintf(desuma, '\n***** RETE 10 *****\n');
run('C:\Users\Laura\Desktop\Tesi\reti\modello\rete10.m')
fprintf(desuma, '\n m = 3, l = 2, d = 1 \n');
!pluginP9.exe "rete10.m" "Output_Files\rete10_matlab_disc.m"
!pluginP2.exe "Output_Files\rete10_matlab_disc.m" "Output_Files\rete10_pnml.pnml"
!adapterA2.exe "Output_Files\rete10_pnml.pnml" "Output_Files\rete10_p3.xml"
tic
!adapterA15_Pipe3toPipe2.exe "Output_Files\rete10_p3.xml"
"Output_Files\rete10_p2.xml"
tic
!tina -R -PNML "Output_Files\rete10_p2.xml" "Output_Files\reach_graph_10.txt"
!pluginP7.exe "Output_Files\reach_graph_10.txt" "Output_Files\reach_graph_10.fsm"
t_R_10 = toc;
fprintf(desuma,'TIME_R = %d\n',t_R_10);
tic
!diag_UR.exe "Output_Files\reach_graph_10.fsm" failure_10.ft
"Output_Files\diagnoser_10.diag" "Output_Files\diagnoser_10.fsm" 2
t_diag_10 = toc;
fprintf(desuma,'TIME_DIAG = %d\n',t_diag_10);
tic
!dcycle.exe "Output_Files\reach_graph_10.fsm" failure_10.ft
"Output_Files\inputFile_10.cycles"
t_cicli_10 = toc;
fprintf(desuma,'TIME_CICLI = %d\n',t_cicli_10);
t_UMDES_10 = toc;
fprintf(desuma,'TIME_UMDES = %d\n',t_UMDES_10);

fprintf(desuma, '\n***** RETE 11 *****\n');
run('C:\Users\Laura\Desktop\Tesi\reti\modello\rete11.m')
fprintf(desuma, '\n m = 3, l = 2, d = 0 \n');
```

```
!pluginP9.exe "rete11.m" "Output_Files\rete11_matlab_disc.m"
!pluginP2.exe "Output_Files\rete11_matlab_disc.m" "Output_Files\rete11_pnml.pnml"
!adapterA2.exe "Output_Files\rete11_pnml.pnml" "Output_Files\rete11_p3.xml"
tic
!adapterA15_Pipe3toPipe2.exe "Output_Files\rete11_p3.xml"
"Output_Files\rete11_p2.xml"
tic
!tina -R -PNML "Output_Files\rete11_p2.xml" "Output_Files\reach_graph_11.txt"
!pluginP7.exe "Output_Files\reach_graph_11.txt" "Output_Files\reach_graph_11.fsm"
t_R_11 = toc;
fprintf(desuma,'TIME_R = %d\n',t_R_11);
tic
!diag_UR.exe "Output_Files\reach_graph_11.fsm" failure_11.ft
"Output_Files\diagnoser_11.diag" "Output_Files\diagnoser_11.fsm" 2
t_diag_11 = toc;
fprintf(desuma,'TIME_DIAG = %d\n',t_diag_11);
tic
!dcycle.exe "Output_Files\reach_graph_11.fsm" failure_11.ft
"Output_Files\inputFile_11.cycles"
t_cicli_11 = toc;
fprintf(desuma,'TIME_CICLI = %d\n',t_cicli_11);
t_UMDES_11 = toc;
fprintf(desuma,'TIME_UMDES = %d\n',t_UMDES_11);

fprintf(desuma, '\n***** RETE 12 *****\n');
run('C:\Users\Laura\Desktop\Tesi\reti\modello\rete12.m')
fprintf(desuma, '\n m = 3, l = 3, d = 1 \n'); !pluginP9.exe
"rete12.m" "Output_Files\rete12_matlab_disc.m" !pluginP2.exe
"Output_Files\rete12_matlab_disc.m" "Output_Files\rete12_pnml.pnml"
!adapterA2.exe "Output_Files\rete12_pnml.pnml"
"Output_Files\rete12_p3.xml" tic !adapterA15_Pipe3toPipe2.exe
"Output_Files\rete12_p3.xml" "Output_Files\rete12_p2.xml" tic !tina
-R -PNML "Output_Files\rete12_p2.xml"
"Output_Files\reach_graph_12.txt" !pluginP7.exe
"Output_Files\reach_graph_12.txt" "Output_Files\reach_graph_12.fsm"
t_R_12 = toc;
fprintf(desuma,'TIME_R = %d\n',t_R_12);
tic
!diag_UR.exe "Output_Files\reach_graph_12.fsm" failure_12.ft
"Output_Files\diagnoser_12.diag" "Output_Files\diagnoser_12.fsm" 2
t_diag_12 = toc;
fprintf(desuma,'TIME_DIAG = %d\n',t_diag_12);
tic
!dcycle.exe "Output_Files\reach_graph_12.fsm" failure_12.ft
"Output_Files\inputFile_12.cycles"
t_cicli_12 = toc;
fprintf(desuma,'TIME_CICLI = %d\n',t_cicli_12);
t_UMDES_12 = toc;
fprintf(desuma,'TIME_UMDES = %d\n',t_UMDES_12);
fclose(desuma);
```

A.4.2 script_tools

```
clear all
pn_diag = fopen('C:\Users\Laura\Desktop\Tesi\reti\modello\
                Output_Files\model_PN_DIAG.txt','wt');
if (pn_diag < 0)
   error('could not open file "C:\Users\Laura\Desktop\Tesi\reti\
          modello\Output_Files\model_PN_DIAG.txt"');
```

```
end
pn_diag = fopen('C:\Users\Laura\Desktop\Tesi\reti\
                    modello\Output_Files\model_PN_DIAG_2.txt','wt');
if (pn_diag_2_tool < 0)
    error('could not open file "C:\Users\Laura\Desktop\Tesi\reti\
           modello\Output_Files\model_PN_DIAG_2_Tool.txt"');
end
for m = 2:3
    for l = 1:3
        for d = 0:1
            run('C:\Users\Laura\Desktop\Tesi\reti\modello\simulation.m')
            fprintf('\n********** TOOLBOX PN_DIAG **********\n');
            fprintf(pn_diag, '\n m = %d, l = %d, d = %d \n', m, l, d);
            run('C:\Akhela\WP4SoftwarePlatform\Tools\PN_DIAG\main_PN_DIAG.m')
            clear brg brd mbrg
            fprintf('\n********** TOOLBOX PN_DIAG_2_Tool **********\n');
            fprintf(pn_diag_2, '\n m = %d, l = %d, d = %d \n', m, l, d);
            run('C:\Akhela\WP4SoftwarePlatform\Tools\PN_DIAG_2\launch_DISC.m')
            clear mbrd mbrg
        end
    end
end
fclose(pn_diag);
fclose(pn_diag_2);
```

Bibliografia

[1] DISC: Distributed supervisory control of complex plants. *http://www.disc-project.eu*. [cited at p. 4, 5, 53]

[2] M. Pocci Home's Page. *http://www.diee.unica.it/giua/TESI/09_Marco.Pocci/*. [cited at p. 2]

[3] Platform Independent Petri net Editor 2. *http://pipe2.sourceforge.net/*. [cited at p. 57]

[4] R. Perria Home's Page. *http://www.diee.unica.it/giua/TESI/11_Rita.Perria/*. [cited at p. 2]

[5] TINA: TIme petri Net Analyzer *http://homepages.laas.fr/bernard/tina//*. [cited at p. 57]

[6] A. Di Febbraro A. Giua. *Sistemi ad eventi discreti*. The McGraw-Hill Companies, 2002. [cited at p. 7]

[7] A. Aghasaryan, E. Fabre, A. Benveniste, R. Boubour, and C. Jard. Fault detection and diagnosis in distributed systems: an approach by partially stochastic Petri nets. *Discrete Events Dynamical Systems*, 8:203–231, June 1998. [cited at p. 3]

[8] F. Basile, P. Chiacchio, and G. De Tommasi. An efficient approach for online diagnosis of discrete event systems. *IEEE Trans. on Automatic Control*. [cited at p. 3]

[9] A. Benveniste, E. Fabre, S. Haar, and C. Jard. Diagnosis of asynchronous discrete event systems, a net unfolding approach. *IEEE Trans. on Automatic Control*, 48(5):714–727, May 2003. [cited at p. 3]

[10] R.K. Boel and G. Jiroveanu. Distributed contextual diagnosis for very large systems. In *Proc. IFAC WODES'04: 7th Work. on Discrete Event Systems*, pages 343–348, September 2004. [cited at p. 3]

[11] R.K. Boel and J.H. van Schuppen. Decentralized failure diagnosis for discrete-event systems with costly communication between diagnosers. In *Proc. WODES'02: 6th Work. on Discrete Event Systems*, pages 175–181, October 2002. [cited at p. 3]

[12] M.P. Cabasino, A. Giua, S. Lafortune, and C. Seatzu. Diagnosability analysis of unbounded Petri nets. In *Proc. 48th IEEE Conf. on Decision and Control*, December 2009. [cited at p. 3]

[13] M.P. Cabasino, A. Giua, M. Pocci, and C. Seatzu. Discrete event diagnosis using labeled Petri nets. An application to manufacturing systems. *Control Engineering Practice*, 2010. doi:10.1016/j.conengprac.2010.12.010. In press. [cited at p. 2, 3, 35, 37, 38, 39, 41, 42, 46, 47, 69, 70]

[14] M.P. Cabasino, A. Giua, and C. Seatzu. Fault detection for discrete event systems using Petri nets with unobservable transitions. *Automatica*, 46(9):1531–1539, 2010. [cited at p. 3, 35, 37]

[15] S. Lafortune C.G. Cassandras. *Introduction to discrete event systems, Second Edition*. Springer, 2007. [cited at p. 10, 24]

[16] S.L. Chung. Diagnosing pn-based models with partial observable transitions. *International Journal of Computer Integrated Manufacturing*, 12 (2):158–169, 2005. [cited at p. 3]

[17] R. Debouk, S. Lafortune, and D. Teneketzis. Coordinated decentralized protocols for failure diagnosis of discrete-event systems. *Discrete Events Dynamical Systems*, 10(1):33–86, January 2000. [cited at p. 3]

[18] S. Genc and S. Lafortune. Distributed diagnosis of discrete event systems using Petri nets. In *Proc. of the 24th ATPN*, pages 316–336, June 2003. [cited at p. 3]

[19] A. Giua and C. Seatzu. Fault detection for discrete event systems using Petri nets with unobservable transitions. In *Proc. 44th IEEE Conf. on Decision and Control*, pages 6323–6328, December 2005. [cited at p. 3]

[20] C.H. Papadimitriou H.R. Lewis. *Elements of the theory of computation*. Prentice-Hall, 1981. [cited at p. 10]

[21] S. Jiang and R. Kumar. Failure diagnosis of discrete-event systems with linear-time temporal logic specifications. *IEEE Trans. on Automatic Control*, 49(6):934–945, June 2004. [cited at p. 3]

[22] G. Jiroveanu and R.K. Boel. Contextual analysis of Petri nets for distributed applications. In *16th Int. Symp. on Mathematical Theory of Networks and Systems (Leuven, Belgium)*, July 2004. [cited at p. 3]

[23] D.B. Johnson. Finding all the elementary circuit of a directed graph. *SIAM J. Comput.*, 4:77–84, 1975. [cited at p. 42, 50]

[24] S. Lafortune. Executables of the umdes-lib software library for solaris, linux, mac and windows are publicly available. *http://www.eecs.umich.edu/umdes/toolboxes.html*. [cited at p. 3, 57, 71]

[25] J. Lunze and J. Schroder. Sensor and actuator fault diagnosis of systems with discrete inputs and outputs. *IEEE Trans. Systems, Man and Cybernetics, Part B*, 34(3):1096–1107, April 2004. [cited at p. 3]

[26] R. Perria. Implementazione di un nuovo algoritmo per la diagnosticabilità di reti posto/transizione mediante marcature di base. Master's thesis, Dep. Electric and Electronic Engineering, University of Cagliari, Cagliari, Italy, 2011. (In Italian). [cited at p. 5, 36, 42, 48, 51, 69, 70, 71, 129, 133, 135, 136]

[27] C.A. Petri. *Kommunication mit Automaten*. PhD thesis, Institut fur Instrumentelle Mathematik, Schriften des IIM, No. 3, Bonn, Germany, 1962. [cited at p. 10]

[28] M. Pocci. Un toolbox per la diagnosticabilità di reti posto/transizione. Master's thesis, Dep. Electric and Electronic Engineering, University of Cagliari, Cagliari, Italy, 2009. (In Italian). [cited at p. 5, 36, 42, 43, 44, 45, 47, 69, 70, 71, 133, 135, 136]

[29] M. Sampath, S. Lafortune, and D. Teneketzis. Active diagnosis of discrete-event systems. *IEEE Trans. on Automatic Control*, 43(7):908–929, July 1998. [cited at p. 3]

[30] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Trans. on Automatic Control*, 40 (9):1555–1575, 1995. [cited at p. 3, 129, 135]

[31] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Failure diagnosis using discrete-event models. *IEEE Trans. Control Systems Technology*, 4(2):105–124, 1996. [cited at p. 3]

[32] F. Sessego. Hypens: un simulatore per le reti di petri discrete, continue e ibride. Master's thesis, Dep. Electric and Electronic Engineering, University of Cagliari, Cagliari, Italy, 2007. (In Italian). [cited at p. 95]

[33] R. Tarjan. Enumeration of the elementary circuit of a directed graph. *SIAM J. Comput.*, 2:211–216, 1972. [cited at p. 46, 50]

[34] J.C. Tiernan. An efficient search algorithm to find the elementary circuits of a graph. *Comm. ACM*, 13:722–726, 1970. [cited at p. 50]

[35] T. Ushio, L. Onishi, and K. Okuda. Fault detection based on Petri net models with faulty behaviors. In *Proc. SMC'98: IEEE Int. Conf. on Systems, Man, and Cybernetics (San Diego, CA, USA)*, pages 113–118, October 1998. [cited at p. 3]

[36] Y. Wen and M. Jeng. Diagnosability analysis based on T-invariants of Petri nets. In *Networking, Sensing and Control, 2005. Proceedings, 2005 IEEE.*, pages 371–376, March 2005. [cited at p. 3]

[37] S. H. Zad, R.H. Kwong, and W.M. Wonham. Fault diagnosis in discrete-event systems: framework and model reduction. *IEEE Trans. on Automatic Control*, 48(7):1199–1212, July 2003. [cited at p. 3]