Analisi di reti posto/transizione mediante ordine parziale: sviluppo di un Toolbox Matlab per "unfolding"

Caterina Carboni, Federica Lamon

Dip. di Ing. Elettrica ed Elettronica, Università di Cagliari, Piazza d'Armi, 09123 Cagliari, Italy

email:catecarboni@tiscali.it,federica.lamon@tin.it

Relatore: Prof. Alessandro Giua

Anno accademico 2004-2005

Ringraziamenti

Desideriamo ringraziare tutti coloro che ci hanno offerto il loro supporto nella preparazione di questa Tesi. Un ringraziamento particolare va al Prof. Alessandro Giua, nostro relatore di Tesi, ordinario del dipartimento di Ingegneria Elettrica ed Elettronica dell'Università degli Studi di Cagliari, per la disponibilità dimostrata.

Indice

Sommario **v**

1 Introduzione **1**

2 Reti di Petri posto/transizione **5**

2.1 Struttura delle reti posto/transizione 5

2.2 Marcatura e sistema di rete 7

2.3 Abilitazione e scatto 7

2.4 Strutture elementari 8

2.5 Analisi mediante grafo di raggiungibilità 10

3 Dispiegamento **20**

3.1 Reti di occorrenze 20

3.2 La procedura di dispiegamento 23

3.3 Prefissi finiti 29

3.3.1 Primo ordine di dispiegamento 30

3.3.2 Secondo ordine di dispiegamento 32

3.3.3 Prefisso finito di McMillan 33

4 Principali funzioni del toolbox Matlab **39**

4.1 Funzione unfolding_n 39
4.1.1 Sintassi 39
4.1.2 Descrizione del programma 41
4.1.3 Procedure per la valutazione della concorrenza 46
4.2 Funzione order1 51
4.2.1 Sintassi 51
4.2.2 Descrizione del programma 51
4.3 Funzione order2 54
4.3.1 Sintassi 55
4.3.2 Descrizione del programma 55
4.4 Funzione McMillan 57
4.4.1 Sintassi 57
4.4.2 Descrizione del programma 59
4.5 Funzioni unfolding_graph,graph1, graph2, graphMcM 60
4.5.1 Sintassi 60
4.6 Funzione reach_u 63
4.6.1 Sintassi 63
4.6.2 Descrizione del programma 64

5 Analisi dei risultati raggiunti **71**

5.1 I filosofi cinesi 71
5.2 Slotted ring protocol 80

A Glossario **81**

B Listati dei programmi **83**

B.1 Funzione unfolding_n 83

B.2 Funzione order1 89

B.3 Funzione order2 96

B.4 McMillan 102

B.5 Funzione unfolding_graph 109

B.6 Funzione graph1 111

B.7 Funzione graph2 113

B.8 Funzione graphMcM 116

B.9 Funzione reach_u 118

B.10 Funzione Filosofi 133

B.11 Funzioni ausiliarie 136

B.11.1 Funzione is_cutoff1 136

B.11.2 Funzione is_cutoff2 137

B.11.3 Funzione firing_seq 139

B.11.4 Funzione is_cutoffMcm 139

B.11.5 Funzione lc_McMillan 141

B.11.6 Funzione previous 142

Bibliografia **143**

Sommario

L'argomento che sarà trattato in questo lavoro è da collocarsi nell'ambito dei Sistemi ad Eventi Discreti e in particolare fa riferimento alla teoria delle reti di Petri posto/transizione.

La tesi, che si apre con una breve introduzione e un richiamo sui concetti di base delle reti di Petri, si concentra su un'importante tecnica di analisi che si avvale della procedura di *dispiegamento* (*unfolding*). Tale metodo consiste nel convertire una qualunque rete di Petri posto/transizione sana e ordinaria in una particolare rete, detta rete di dispiegamento, appartenente alla classe delle reti di occorrenza. Essa è una rete di Petri di lunghezza infinita dalla quale si possono ricavare dei prefissi finiti tramite delle modifiche introdotte sulla procedura di base.

Lo scopo principale di questo lavoro è stato quello di implementare un toolbox Matlab come strumento di supporto per la costruzione della rete di dispiegamento e dei suoi prefissi finiti.

Le prestazioni raggiunte dagli algoritmi implementati sono state messe a confronto con quelle ottenute utilizzando la più classica tecnica di analisi fondata sul *grafo di raggiungibilità*. Tale confronto è stato portato avanti sia sulla base dei tempi di computazione sia sulla base del numero di stati raggiunti. I risultati ottenuti mostrano come, sotto entrambe le prospettive, l'analisi che si basa sul dispiegamento offra prestazioni decisamente più soddisfacenti rispetto all'analisi tradizionale.

Capitolo 1

Introduzione

Le reti di Petri posto/transizione furono introdotte per la prima volta da Carl Adam Petri, un ricercatore tedesco, che nel 1962 discusse la sua tesi di dottorato dal titolo 'Kommunication mit Automaten', in cui presentava questo nuovo modello che in seguito avrebbe appunto preso il suo nome.

Esse rappresentano uno degli strumenti più efficaci nell'analisi di tutto quel ramo dell'automatica che si occupa di studiare i sistemi ad eventi discreti (SED). Con questo termine si fa riferimento a quei sistemi il cui comportamento può essere rappresentato mediante le sequenze di eventi che essi generano. Esistono due modelli di sistemi ad eventi discreti: modelli temporizzati e modelli logici. I modelli temporizzati consentono di rappresentare la temporizzazione degli eventi mentre quelli logici rappresentano solo l'ordine con cui essi si verificano. Nel seguito si tratteranno unicamente i modelli logici.

I fattori che determinano la particolare importanza assunta dalle reti di Petri sono molteplici: innanzitutto si può dire che le reti di Petri hanno il duplice vantaggio di poter essere rappresentate sia con un semplice formalismo grafico, che in forma matriciale. Ciò consente quindi il loro utilizzo sia come aiuto visuale in sede di progetto sia come strumento matematico per l'analisi e lo studio di varie proprietà. Esse permettono inoltre una rappresentazione compatta anche per sistemi con grandi spazi di stato; questo è possibile in quanto una rete di Petri non enumera tutti gli stati che può assumere un sistema, ma rappresenta solamente le regole che ne determinano l'evoluzione.

Un aspetto che le reti di Petri si prestano a rappresentare in maniera particolarmente efficiente, del quale si parlerà più approfonditamente nel seguito, è la concorrenza, cioè la rappresentazione di attività che possono venir svolte le une indipendentemente dalle altre, in modo completamente parallelo. Altro grosso vantaggio è la possibilità di strutturare un sistema in maniera modulare. Ciò consente di poter pensare il sistema collettivo come un in-

sieme di sottoreti che possono così essere progettate singolarmente, e che poi possono essere collegate tra loro in maniera abbastanza semplice.

La principale tecnica di analisi che consente di studiare le proprietà di una rete si avvale della costruzione del *grafo di raggiungibilità*. Esso è uno strumento che pevmette di rappresentare tutti gli stati in cui può trovarsi un qualsiasi sistema. Negli ultimi 10-15 anni, ha assunto nella letteratura scientifica internazionale una rilevanza sempre maggiore un'originale tecnica di analisi basata su un nuovo modo di rappresentare la rete. Essa si avvale non più del grafo di raggiungibilità, ma di una particolare classe di reti che segue un modello ad ordine parziale[1]. Queste prendono il nome di *reti di occorrenze* (*occurence net*[2]) . La conversione di un rete di Petri qualunque in una rete appartenente a tale classe è sempre possibile, e la tecnica che si utilizza viene chiamata *dispiegamento* (*unfolding*[3]). La procedura di dispiegamento genera, a partire da una rete marcata ordinaria e sana $< N, M_0 >$, una rete di occorrenze che sarà chiamata appunto "dispiegamento". Il dispiegamento descrive in termini strutturali, le possibili evoluzioni della rete di partenza esplicitando tutti i possibili eventi (scatti di transizioni) e le relazioni fra loro. Vari autori si sono già occupati di questi argomenti: in questa tesi verrà focalizzata l'attenzione sul lavoro portato avanti da Giua e Xie (2005).

Il vantaggio nell'utilizzo di questa nuova tecnica trova riscontro sia dal punto di vista dell'efficienza computazionale che dal punto di vista della compattezza nella rappresentazione grafica. Tali benefici si apprezzano maggiormente in tutte quelle reti che hanno nella loro conformazione una forte componente di concorrenzialità. E' noto infatti che un'analisi basata sull'utilizzo del grafo di raggiungibilità rende obbligatorio l'elenco di tutte le marcature raggiungibili. Il problema si pone quando si ha a che fare con delle reti con un alto numero di componenti concorrenti in quanto il numero delle marcature cresce in maniera esponenziale con il numero di tali componenti. Tale problema, meglio noto come esplosione dello spazio di stato, produce risultati evidentemente poco pratici anche perché non sempre è necessario conoscere tutte le marcature. Questo problema non si pone utilizzando il dispiegamento perché le marcature raggiungibili sono già rappresentate in maniera assai più compatta nella struttura stessa della rete.

Come si è già detto la procedura di dispiegamento genera una rete di occorrenze. Con questo termine si indica una rete di Petri aciclica, che non contiene cioè cicli orientati. La caratteristica principale di una rete di occorrenze, da cui derivano i vantaggi di cui si è appena parlato, consiste nell'esplicitare tutte le possibili evoluzioni della rete. In questo modo si ottiene una

[1] Si veda in Appendice A la definizione di "ordine parziale".

[2] Una traduzione potrebbe essere rete di eventi. Tale scelta e' la piu' elegante. Pero' per consistenza con il termine inglese, tenendo conto che "occorrenza" ha ormai assunto un particolare significato tecnico, si è deciso di tradurlo con rete di occorrenze. Si veda in Appendice A il significato di tale termine.

[3] Le traduzioni più adatte sono due: spiegamento e dispiegamento. E' stato scelto il secondo termine in quanto di uso meno comune e quindi di minore ambiguità. Inoltre, come riportato in Appendice A, il significato di tale termine risulta particolarmente adatto a descrivere le caratteristiche delle reti in questione.

rete un po' più grande dell'originale ma capace di rappresentare in maniera simultanea e compatta tutti i percorsi concorrenti.

L'algoritmo di dispiegamento di per se conduce a una rete di Petri di lunghezza infinita in quanto converte i cicli che erano presenti nella rete di partenza in percorsi lineari. Tale procedura può essere replicata infinite volte: è infatti chiaro che se nella rete originale è presente un ciclo, questo può essere percorso un numero qualsiasi di volte. In realtà si può dimostrare che esiste un prefisso finito della rete di dispiegamento che contiene tutti e soli i posti della rete necessari per rappresentare in modo completo lo spazio di stato.

Questo lavoro si basa sullo sviluppo di un toolbox Matlab (©1984-2002, The MathWorks, Inc.) in cui sono stati implementati sia l'algoritmo di dispiegamento e sia altri algoritmi utili per l'analisi di alcune proprietà delle reti. I risultati, verificati tramite dei test, dimostrano che l'efficienza computazionale dell'analisi mediante dispiegamento supera di gran lunga quella che si avvale del grafo di raggiungibilità.

Nel Capitolo 2 si parlerà delle reti di Petri posto/transizione nella loro generalità, descrivendone la struttura, le regole che governano la loro evoluzione e i parametri che le caratterizzano. Verrà quindi presentata un'importante tecnica di analisi che si avvale della costruzione del grafo di raggiungibilità, e si parlerà del problema dell'esplosione dello spazio di stato.

Nel Capitolo 3 si analizzerà più in dettaglio quello che è il tema di fondo trattato nella tesi: verrà fornita la definizione di "rete di occorrenza" e di "procedura di dispiegamento". Sarà quindi descritto l'algoritmo utilizzato per generare il dispiegamento. In tale capitolo si spiegherà inoltre cosa si intende per prefisso finito e verrà fatta una distinzione tra prefisso finito di McMillan, primo ordine di dispiegamento, secondo ordine di dispiegamento e saranno presentati i rispettivi algoritmi.

Nel Capitolo 4 saranno illustrate le funzioni che compongono il pacchetto Matlab. Innanzitutto verrà descritta l'implementazione della procedura generale di dispiegamento per poi entrare più nello specifico e distinguere i diversi casi a seconda del prefisso finito che si intende utilizzare. Infine forniremo alcune *utilities* volte a facilitare l'analisi delle reti di Petri mediante l'utilizzo del dispiegamento. Verranno inoltre messi in evidenza i problemi sorti nell'implementazione diretta dell'algoritmo così come esso è stato scritto e le soluzioni adottate per far fronte a tali difficoltà.

Infine, il Capitolo 5 è dedicato a un confronto tra i risultati raggiunti con i due diversi metodi di analisi (grafo di raggiungibilità e dispiegamento) sia sulla base dei tempi di processamento che sulla base del numero di stati ottenuti.

In Appendice A viene riportato un piccolo glossario in cui verranno discusse le scelte che sono state fatte nel tradurre alcune parole chiave della tesi dall'inglese all'italiano, mentre nell'Appendice B sono riportati tutti i listati creati per lo sviluppo del software.

Infine, si ricorda che la pagina web di questa tesi può essere consultata all'indirizzo `http://www.diee.unica.it/automatica/tesi/CarboniLamon`. In tale pagina sarà possibile scaricare il Toolbox Matlab sul dispiegamento corredato da un sintetico manuale d'utilizzo, e vi saranno dei collegamenti ad altre risorse sull'argomento disponibili in rete.

Capitolo 2

Reti di Petri posto/transizione

2.1 Struttura delle reti posto/transizione

In questo capitolo viene descritto il modello di reti di Petri più semplice, detto rete posto/transizione (o rete P/T). Si tratta di un modello logico, che non consente di rappresentare la temporizzazione degli eventi ma solo l'ordine con cui essi si verificano.

Nel seguito verrà descritta la struttura algebrica e grafica delle reti di Petri posto/transizione.

Una rete di Petri P/T è un grafo bipartito, orientato e pesato. I due tipi di vertici sono detti posti (rappresentati da cerchi) e transizioni (rappresentati da barre). Gli archi, che devono essere orientati, connettono i posti alle transizioni e viceversa.

Definizione 2.1. *Una rete P/T è una struttura* $N = (P, T, Pre, Post)$ *dove:*

P *: è l'insieme degli* m *posti.*

T*: è l'insieme delle* n *transizioni.*

Pre*: è la funzione di pre-incidenza che specifica gli archi diretti dai posti alle transizioni (detti archi pre) e viene rappresentata mediante una matrice* $m \times n$*.*

$Post$*: è funzione di post-incidenza che specifica gli archi diretti dalle transizioni ai posti (detti archi post) e viene rappresentata mediante una matrice* $m \times n$*.*

L'informazione della struttura di rete contenuta nelle matrici Pre, $Post$ può essere compattata in un unica matrice, detta di *incidenza*.

Definizione 2.2. *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:*

$$C = Post - Pre$$

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

${}^\bullet t = \{p \in P \mid Pre(p,t) > 0\}$: 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.

Un esempio chiarirà questi concetti:

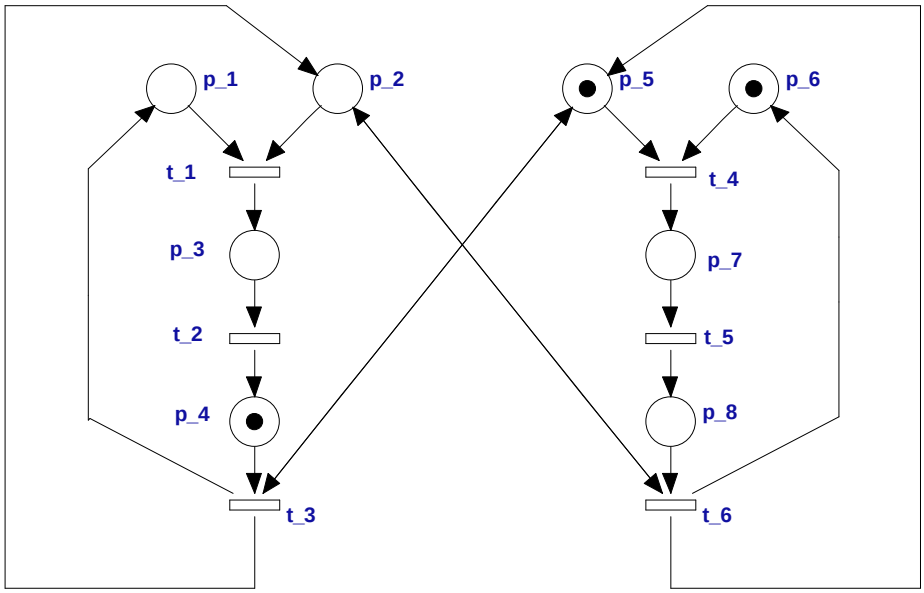

Figura 2.1: Una rete di Petri posto/transizione

Esempio 2.1. Considerando la rete in Figura 2.1 (Giua e Xie, 2005), possono essere scritte le matrici Pre, Post e di incidenza secondo le regole appena descritte:

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

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

Vale inoltre: ${}^\bullet t_1 = \{p\_1, p\_2\}$, $t\_1^\bullet = \{p\_3\}$, ${}^\bullet p\_2 = \{t\_3, t\_6\}$, $p\_2^\bullet = \{t\_1, t\_6\}$.

L'arco bi-orientato viene usato per una migliore chiarezza grafica. Esso rappresenta in verità due archi: uno in ingresso alla transizione e l'altro in uscita. In questo esempio: $t\_3^\bullet = \{p\_1, p\_2, p\_5\}$, ${}^\bullet t\_3 = \{p\_4, p\_5\}$.

$\diamond$

Ora verrà spiegato in che modo è possibile associare un'evoluzione degli stati alla struttura appena descritta:

2.2 Marcatura e sistema di rete

Mediante la marcatura è possibile definire lo stato di una rete P/T. Una marcatura è pertanto una funzione $M : P \longrightarrow N$ che assegna ad ogni posto un numero non negativo di marche (o gettoni) rappresentate graficamente con dei pallini neri dentro i posti. Si indica con $M(p)$ la marcatura associata al generico posto p. Una rete N con marcatura iniziale M_0 è detta *rete marcata* (o sistema di rete) è viene indicata come $\langle N, M_0 \rangle$.

Nella rete in Figura 2.1 la marcatura rappresentata è $M = [\ 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0\ ]$ ossia $M(p\_1) = 0, M(p\_2) = 0, M(p\_3) = 0, M(p\_4) = 1, M(p\_5) = 1, M(p\_6) = 1, M(p\_7) = 0, M(p\_8) = 0$.

2.3 Abilitazione e scatto

Una transizione t è abilitata dalla marcatura M se

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

cioè se ogni posto $p \in P$ dalla rete contiene un numero di marche pari o superiore a $Pre(p, t)$. Se la transizione t è abilitata essa può *scattare*. Lo scatto di t rimuove $Pre(p, t)$ marche da ogni posto $p \in P$ e ne aggiunge $Post(p, t)$ in ogni posto $p \in P$, determinando una nuova marcatura M'. Cioè vale:

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

Per indicare che lo scatto di t da M determina la marcatura M' si scrive $M[t\rangle M'$. Per indicare invece che una sequenza di transizioni $\sigma = t\_1, t\_2 \ldots t_k$, detta anche *sequenza di scatto*, è abilitata a partire da M si scrive $M[\sigma\rangle M'$.

Definizione 2.3. *Data una sequenza di scatto σ si associa ad essa un* vettore di scatto $X : T \longrightarrow \mathbb{N}$ *tale che $X(t) = K$ se la transizione t compare K volte in σ.*

Esempio 2.2. Nella rete in figura 3.4, si consideri la sequenza $\sigma = t\_3, t\_1, t\_2, t\_3$. Poichè t_3 compare 2 volte nella sequenza di scatto σ, mentre t_1, t_2 solo una volta e t_4, t_5, t_6 non compaiono neanche una volta, il vettore di scatto di tale sequenza sarà $X = [112000]$. $\diamond$

Una marcatura M è detta *morta* se nessuna transizione è abilitata da M.

Una marcatura M è detta *raggiungibile* in $\langle N, M_0\rangle$ se esiste una sequenza di scatto tale che $M_0[\sigma\rangle M$. L'insieme di marcature raggiungibili da M_0 definisce l'insieme di raggiungibilità di $\langle N, M_0\rangle$ e si denota con $R(N, M_0)$.

Una rete posto/transizione è detta *ordinaria* se $Pre : P \times T \longrightarrow \{0, 1\}$ e $Post : P \times T \longrightarrow \{0, 1\}$, cioè se ogni arco ha molteplicità unitaria.

Un posto p è detto *k-limitato* in $\langle N, M_0\rangle$ se per ogni marcatura $M \in R(N, M_0)$ vale $M(p) \leq k$. Un posto 1-limitato viene detto *sano*. Una rete marcata è k-limitata se tutti i suoi posti sono k-limitati. Una rete marcata 1-limitata è detta *sana*. Nel seguito, per semplicità, verranno trattate solo sistemi di reti sane.

2.4 Strutture elementari

In un sistema ad eventi discreti, l'ordine con cui possono verificarsi i diversi eventi è soggetto a vincoli di diversa natura. In una rete di Petri P/T ciò corrisponde a porre dei vincoli sull'ordine con cui scattano le transizioni.

E' possibile individuare quattro principali strutture elementari:

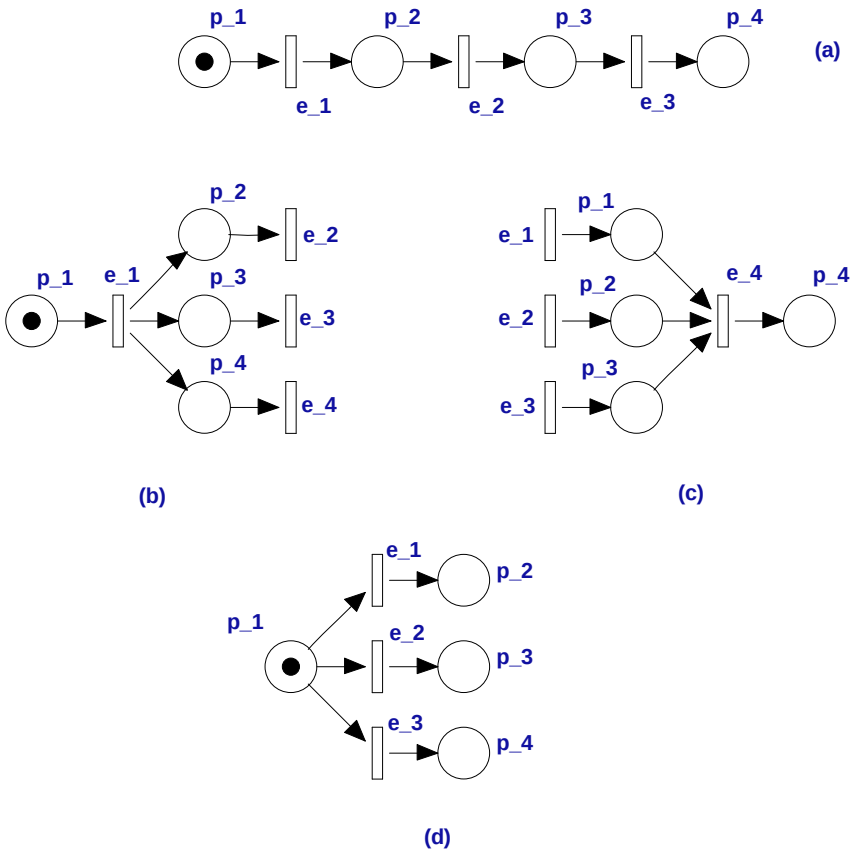

Figura 2.2: Strutture elementari di reti di Petri: (a) Sequenzialità; (b) Parallelismo; (c) Sincronizzazione; (d) Scelta.

Sequenzialità: Gli eventi si succedono in un determinato ordine.

Esempio 2.3. In Figura 2.2 l'evento e_2 può verificarsi (scattare) solo dopo che si è verificato l'evento e_1 e via di seguito. ◇

Parallelismo (**o concorrenza strutturale**): Gli eventi possono verificarsi senza alcun ordine prefissato.

Esempio 2.4. La Figura 2.2 mostra che dopo che si verifica l'evento e_1 i posti p_2, p_3, p_4 vengono marcati contemporaneamente e gli eventi e_2, e_3, e_4 sono simultaneamente abilitati. ◇

Sincronizzazione: Più eventi paralleli devono verificarsi per poter procedere.

Esempio 2.5. La Figura 2.2 mostra che l'evento e_4 non può verificarsi se i posti p_1, p_2, p_3 non sono tutti e tre marcati. ◇

Scelta (**o conflitto strutturale**): Un solo evento tra tanti possibili può verificarsi.

Esempio 2.6. In Figura 2.2 (Di Febbraro e Giua, 2002), solo uno fra gli eventi e_1, e_2, e_3 può verificarsi, perchè il verificarsi di uno qualsiasi degli eventi non permette il verificarsi degli altri eventi in quanto le transizioni corrispondenti non possono essere abilitate. ◇

2.5 Analisi mediante grafo di raggiungibilità

Il grafo di raggiungibilità di una rete marcata è un grafo in cui ogni nodo corrisponde ad una marcatura raggiungibile, e ad ogni arco corrisponde una transizione. La tecnica di analisi basata su tale grafo è di tipo esaustivo, nel senso che richiede di enumerare l'intero spazio di stato, ossia tutte le marcature raggiungibili dal sistema. Se la rete marcata ha uno spazio di stato infinito allora anche il suo grafo di raggiungibilità sarà infinito. In tal caso si costruisce il grafo di copertura che non verrà trattato in questa occasione. Per approfondimenti si rimanda a *Sistemi ad eventi discreti* (Di Febbraro e Giua 2002).

Algoritmo 2.1 (**Grafo di raggiungibiltà**). *Il grafo di raggiungibilità di una rete marcata $\langle N, M_0 \rangle$ con matrice di incidenza C si costruisce con la seguente procedura:*

1. *Il nodo radice è la marcatura iniziale M_0. Questo nodo non è inizialmente etichettato.*

2. *Si consideri un nodo M del grafo senza etichetta.*

   (a) *Per ogni transizione t abilitata da M, cioè tale che $M \geq Pre(\cdot, t)$:*

      i. Si calcoli la marcatura $M' = M + C(\cdot, t)$ raggiunta da M allo scatto di t.

      ii. Se non esiste già un nodo M' nel grafo, si aggiunga il nodo M' al grafo.

      iii. Si aggiunga un arco t tra M ed M'.

   (b) *Si etichetti il nodo M vecchio.*

3. *Se esistono nodi senza etichetta si ritorni a* 2.

4. *Si rimuovano tutte le etichette dai nodi.*

Tale procedura termina solo dopo un numero finito di passi e solo se la rete marcata ha un insieme di raggiungibilità finito. In generale il passo 2 dell'algoritmo si ripete per ogni marcatura raggiungibile.

Esempio 2.7. In Figura 2.3 viene costruito il grafo di raggiungibilità della rete in Figura 2.1. ◇

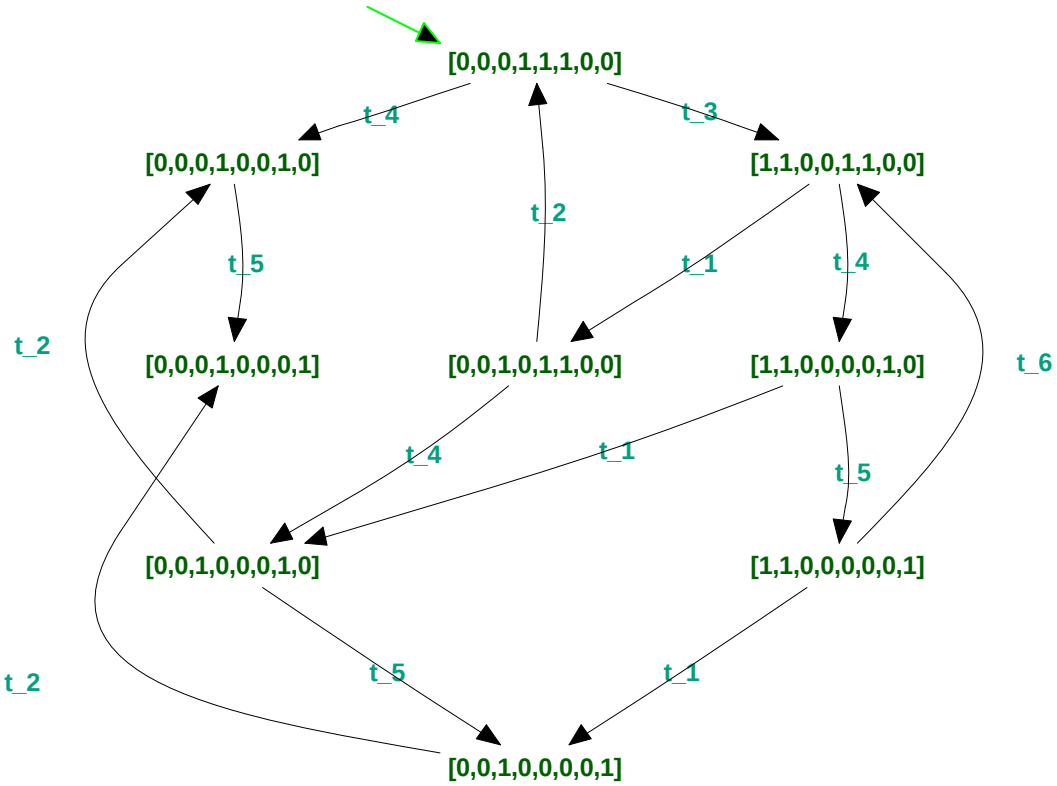

Figura 2.3: Grafo di raggiungibilità della rete in Figura 2.1.

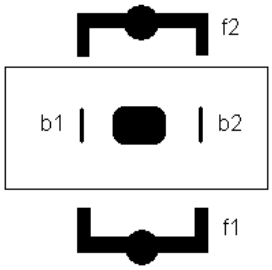

Figura 2.4: I due filosofi cinesi.

L'analisi basata sull'utilizzo del grafo di raggiungibilità obbliga pertanto ad elencare tutte le marcature raggiungibili. Ciò descrive la più grande limitazione di tale metodo di analisi. Il problema si pone infatti quando si ha a che fare con delle reti con un alto numero di componenti concorrenti. Il numero delle marcature infatti cresce in maniera esponenziale con il numero di tali componenti rimanendo tuttavia finito. Tale problema, meglio noto come esplosione dello spazio di stato, si traduce con la realizzazione di un grafo di raggiungibilità di complessità crescente all'aumentare delle marcature.

Per chiarire questo concetto verrà utilizzato un esempio già trattato in precedenza da vari ricercatori in questi ambito.

Esempio 2.8. Due filosofi cinesi (f1, f2) siedono ad un tavolo e meditano. Al centro del tavolo vi è un piatto di riso e tra i due filosofi vi sono 2 bacchette (b1, b2) come mostra la Figura 2.4. Quando un filosofo sente lo stimolo della fame, afferra prima la bacchetta alla sua sinistra, poi l'altra e servendosi della due bacchette mangia. Terminato il pasto, ripone contemporaneamente le bacchete e ritorna alle sue meditazioni.

In Figura 2.5 viene mostrata la rete di Petri posto/transizione del sistema appena descritto.

Nella Tabella 2.1 verrà descritto il significato dei singoli stati della rete rappresentata in Figura 2.5.

Il rispettivo grafo di raggiungibilità, rappresentato in Figura 2.6, mostra che le marcature raggiungibili sono 6.

Se si aumenta il numero degli stati per ciascun filosofo, ad esempio si aggiunge uno stato tra il momento in cui il filosofo impugna la prima bacchetta e il momento in cui impugna la seconda (tale posto può essere associato ad un qualsiasi stato), si potrà vedere come il numero delle marcature cresca esponenzialmente.

La Figura 2.7 rappresenta il caso di due filosofi ciascuno con quattro possibili stati.

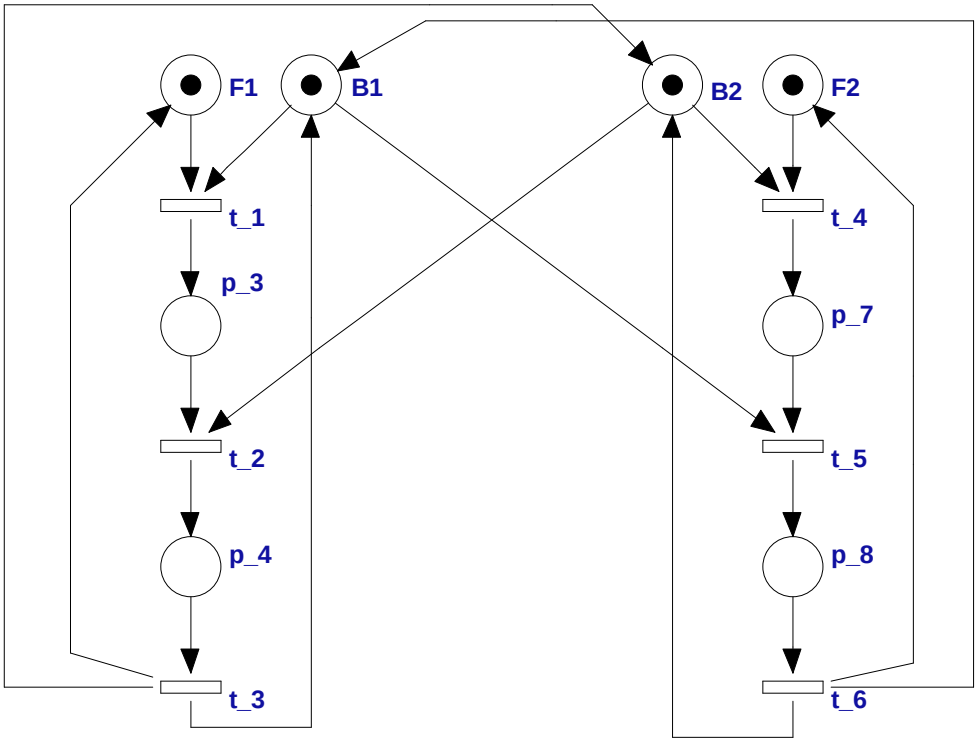

Figura 2.5: Rete posto/transizione equivalente al sistema dei due filosofi cinesi in Figura 2.4.

Posto	**Descrizione**
B1	Bacchetta (b1) sul tavolo, alla sinistra del primo filosofo
B2	Bacchetta (b2) sul tavolo, alla sinistra del secondo filosofo
F1	Stato di meditazione del primo filosofo
F2	Stato di meditazione del secondo filosofo
p_3	Stato in cui il primo filosofo ha afferrato la bacchetta alla sua sinistra
p_7	Stato in cui il secondo filosofo ha afferrato la bacchetta alla sua sinistra
p_4	Stato in cui il il primo filosofo ha afferrato entrambe le bacchette e mangia
p_8	Stato in cui il secondo filosofo ha afferrato entrambe le bacchette e mangia

Tabella 2.1: Significato dei posti della rete in Figura 2.5.

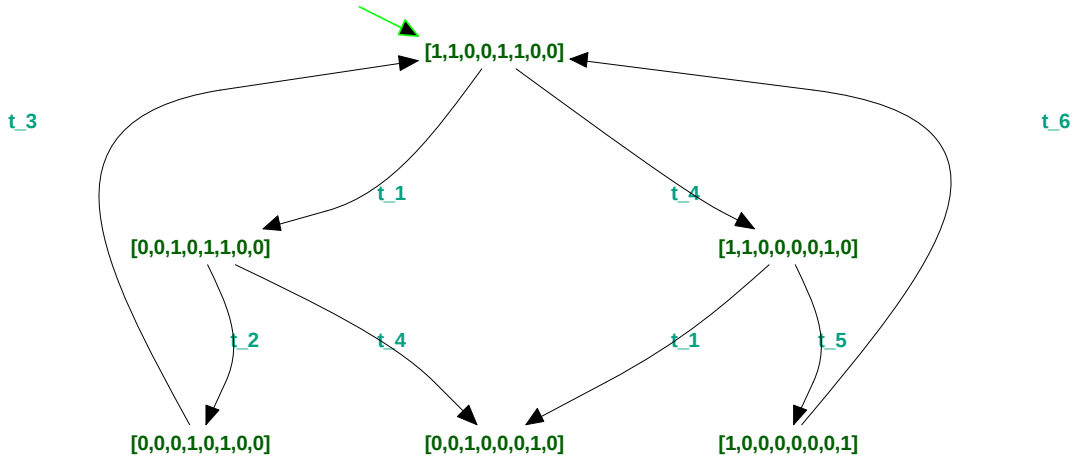

Figura 2.6: Grafo di raggiungibilità della rete in Figura 2.5.

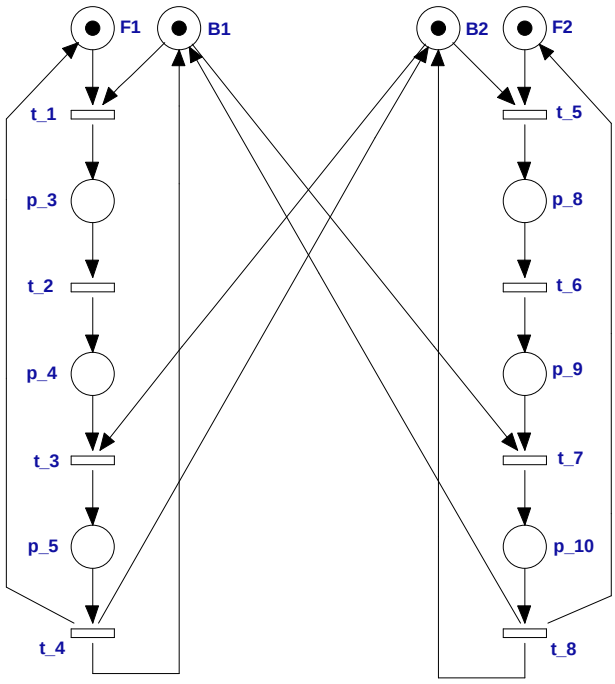

Figura 2.7: Rete posto/transizione equivalente a due filosofi cinesi. Ad ognuno di essi sono associati quattro possibili stati.

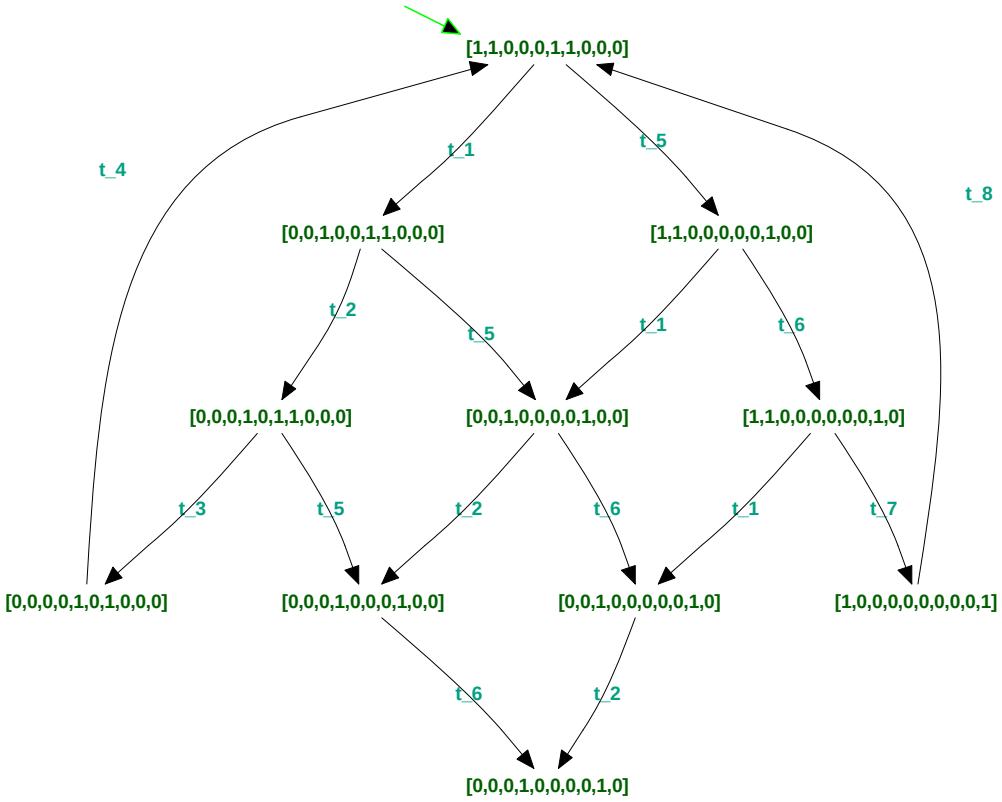

Figura 2.8: Grafo di raggiungibilità della rete in Figura 2.7.

Il grafo di raggiungibilità della rete in Figura 2.7 è mostrato in Figura 2.8. Essa evidenzia come la sola aggiunta di due stati, uno per filosofo, abbia portato al raddoppio delle marcature.

Nella Tabella 2.2 si riporta il numero di marcature al crescere degli stati per ciascun filosofo.

Si supponga ora che siano tre i filosofi seduti allo stesso tavolo e che ad ogni filosofo corrispondano tre diversi stati. La rete di Petri posto/transizione corrispondente è riportata in Figura 2.9.

Il grafo di raggiungibilità corrispondente in Figura 2.10, mostra che le marcature raggiungibili sono 14.

Nella Tabella 2.3 viene fornito il numero di marcature raggiunte dalla rete nel caso in cui sia il numero dei filosofi a variare. Ad ogni filosofo sono associati sempre tre possibili stati.

E' stata creata una funzione Matlab utile per poter visualizzare le matrici Pre, Post e la marcatura iniziale di un qualsiasi sistema di filosofi. La sua interfaccia è:

```
[Pre,Post,M0]=Filosofi(n,m)
```

stati per filosofo	marcature raggiungibili
3	6
4	11
5	18
6	27
7	38
8	51

Tabella 2.2: Marcature raggiungibili per varie possibilità di stato per ciascuno dei due filosofi.

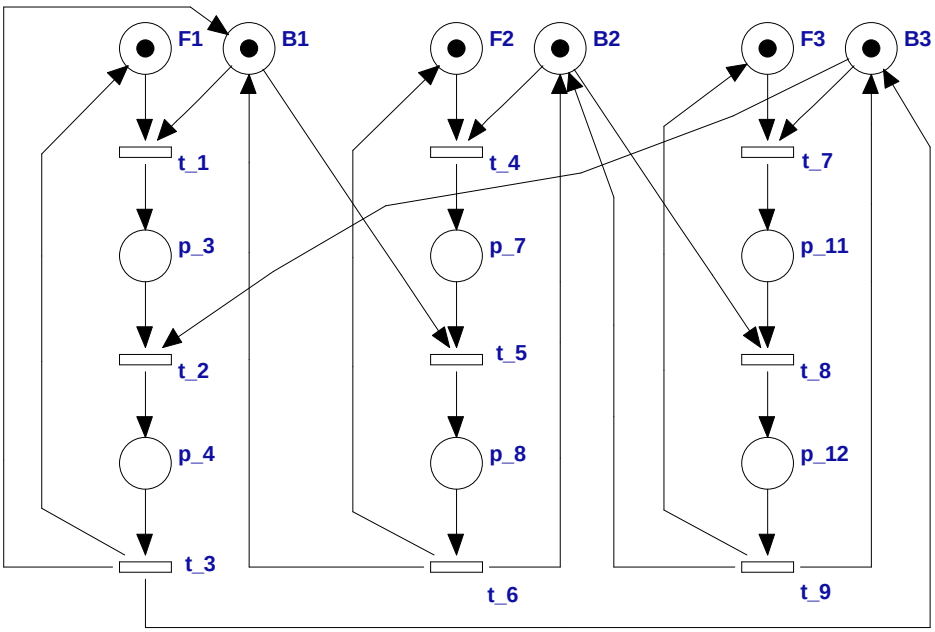

Figura 2.9: Rete posto/transizione equivalente a un sistema con tre filosofi cinesi, con tre stati ciascuno.

numero di filosofi	marcature raggiungibili
2	6
3	14
4	34
5	82
6	..

Tabella 2.3: Marcature raggiungibili per un numero variabile di filosofi seduti allo stesso tavolo. Il simbolo $(..)$ indica che i tempi computazionali necessari all'ottenimento di quel risultato sono eccessivamente elevati.

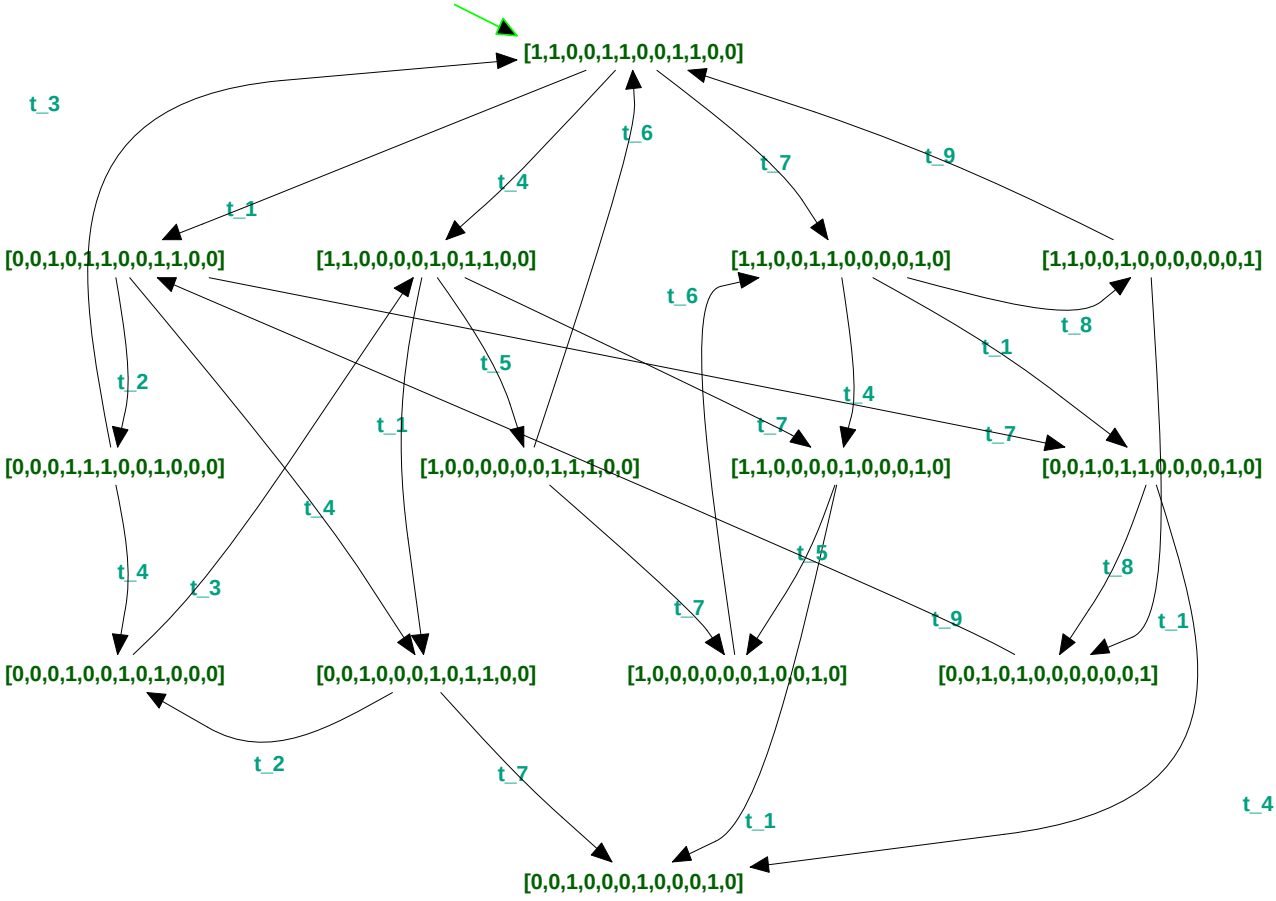

Figura 2.10: Grafo di raggiungibilità per la rete in Figura 2.9.

Tale funzione riceve in ingresso il parametro `n` che indica il numero dei filosofi e il parametro `m` che indica il numero degli stati possibili per ciascun filosofo.

Un esempio di funzionamento è:

```
[Pre,Post,M0]=Filosofi(2,4)
```

Inserendo tali ingressi si intende calcolare quali saranno le matrici Pre, Post e la marcatura inziale di un sistema composto da due filosofi ciascuno con quattro possibili stati.

I dati in uscita saranno:

```
Pre =

     1     0     0     0     0     0     0     0
     1     0     0     0     0     0     1     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     0     0     0
     0     0     1     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

Post =

     0     0     0     1     0     0     0     0
     0     0     0     1     0     0     0     1
     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     0     0     0     0     1
     0     0     0     1     0     0     0     1
     0     0     0     0     1     0     0     0
     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     1     0

M0 =
     1
     1
     0
     0
```

```
0
1
1
0
0
0
```

◇

Capitolo 3

Dispiegamento

Come già accennato in precedenza, con questo termine si indica un particolare modello ad ordine parziale che consente di convertire una rete di Petri sana e ordinaria in una speciale tipologia di reti dette reti di occorrenze.

3.1 Reti di occorrenze

Vengono ricordate brevemente le nozioni utilizzate nel seguito.

Definizione 3.1.

- *Un nodo* x precede x' *se esiste un percorso diretto che porta da* x *a* x'*. Tale relazione si indica come* $x \prec x'$*.*
- *I nodi* x *e* x' *sono in* conflitto *(*$x \# x'$*) se esistono due diverse transizioni* $t, t' \in T$ *tali che* $t \preceq x, t' \preceq x', {}^\bullet t \cap {}^\bullet t' \neq \emptyset$*. Questo infatti significherebbe che uno stesso posto abilita due diverse transizioni e in reti sane, come sono quelle che verranno trattate, non possono mai scattare entrambe. Da qui il conflitto che consente a una sola di esse di poter scattare.*
- *Due nodi invece* x *e* x' *si dicono* concorrenti *se nessuna delle seguenti relazioni è verificata:* $x \preceq x', x' \preceq x, x \# x'$*.*

Sono stati indicati con x e x' due generici nodi della rete di dispiegamento (essi possono cioè rappresentare un posto oppure una transizione).

Esempio 3.1. In Figura 3.1 sono riportati dei semplici esempi delle definizioni appena fornite:

- Figura 3.1 (a): tutti i nodi precedono il nodo p_4, mentre il nodo t_2 è preceduto solo dai nodi p_1, t_1, p_2.
- Figura 3.1 (b): i nodi p_2, p_3, p_4 sono in conflitto per via del posto p_1 che, avendo un solo gettone, può abilitare una sola delle transizioni t_1, t_2, t_3.
- Figura 3.1 (c): i nodi p_2, p_3 e p_4 sono concorrenti, possono infatti essere contemporaneamente marcati in quanto nessuna delle sopraelencate condizioni è verificata.

Si sarà notata una corrispondenza tra le strutture elementari presentate nel Capitolo 2 (Figura 2.2) e le relazioni appena descritte. In effetti la relazione di precedenza corrisponde alla sequenzialità, il conflitto corrisponde alla scelta, mentre la concorrenza corrisponde al parallelismo. ◇

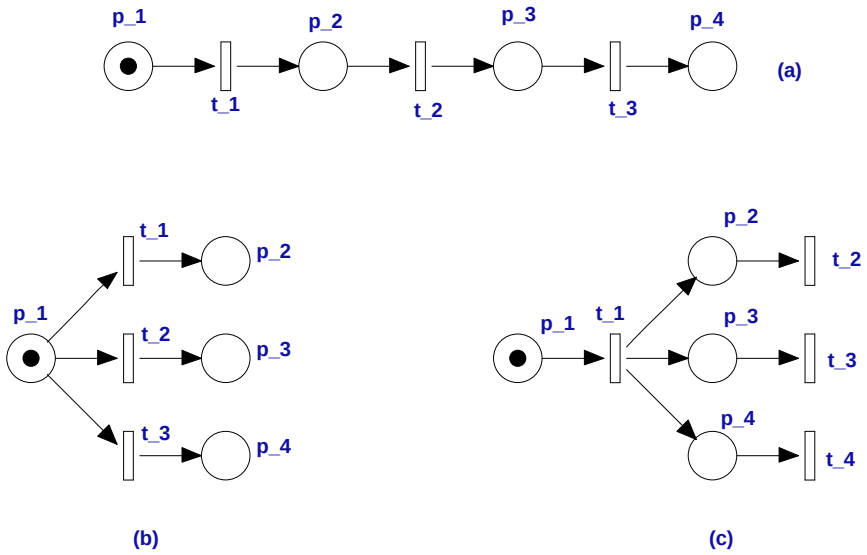

Figura 3.1: (a) Relazione di precedenza, (b) Relazione di conflitto, (c) Relazione di concorrenza.

Definizione 3.2. *Si definisce* rete di occorrenze (occurence net) *una rete di Petri che soddisfa le seguenti tre proprietà:*

- *Partendo da un nodo qualsiasi, tutti i percorsi a ritroso sono finiti, raggiungono cioè un nodo sorgente (ciò determina l'aciclicità della rete).*
- *Ogni posto ha al massimo un arco in ingresso.*
- *Nessun nodo è in conflitto con se stesso.*

Esempio 3.2. In Figura 3.2 è mostrata una rete di occorrenza. Come si può notare essa non contiene nessun ciclo orientato e ogni posto ha un solo arco in ingresso. ◇

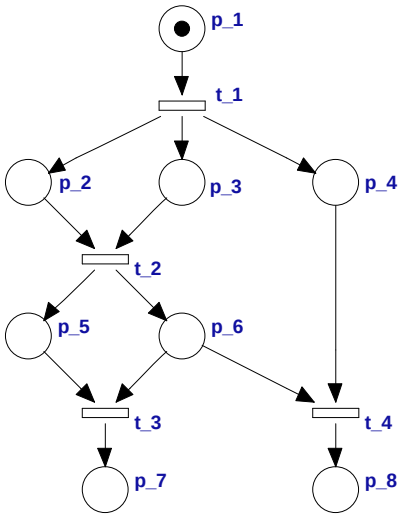

Figura 3.2: Una rete di occorrenze.

In questo tipo di reti si può inoltre dimostrare che le relazioni di precedenza , di conflitto e di concorrenza tra i nodi sono mutuamente esclusive . Dunque una sola delle seguenti relazioni può sussistere: $x \prec x'$, o $x' \prec x$, o $x \# x'$, o $x \approx x'$. Questo non è vero in una rete posto/transizione qualsiasi.

Esempio 3.3. In Figura 3.3 è rappresentata una generica rete di Petri sana e ordinaria. I posti p_2 e p_3, ad esempio, sono in conflitto, infatti, essendo il posto p_1 sano, può scattare solo una tra t_1 e t_3. Gli stessi posti però possono anche essere considerati l'uno precedente all'altro; per rendersene conto è sufficiente percorrere prima il ciclo $\{p\_1, t\_1, p\_2, t\_2, p\_1\}$ e in seguito il ciclo $\{p\_1, t\_3, p\_3, t\_4, p\_1\}$. La presenza dei cicli rende impossibile anche stabilire se per due nodi x e x' della rete valga la relazione $x \prec x'$, oppure $x' \prec x$. In una rete di occorrenze, grazie alla proprietà di aciclicità questo problema non sussiste e le relazioni indicate non possono mai essere vere contemporaneamente. ◇

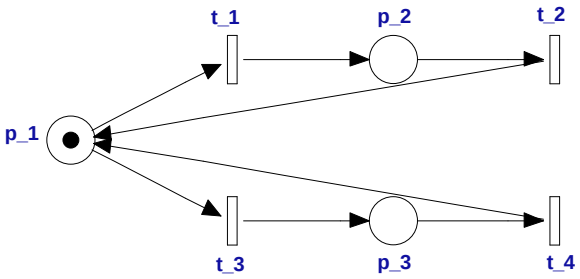

Figura 3.3: Rete posto/transizione.

Le reti di occorrenze hanno il prezioso vantaggio di consentire la rappresentazione simultanea di tutte le evoluzioni che la rete può generare. Questo evita la lunga e spesso inutil-

mente complicata ricerca di tutte le marcature raggiungibili che, come si è detto nel capitolo precedente, conduce in alcune reti al problema dell'esplosione dello spazio di stato.

3.2 La procedura di dispiegamento

In questo paragrafo verrà descritta la metodologia generale che consente di ottenere una rete di occorrenze a partire da una qualunque rete di Petri sana e ordinaria.

Definizione 3.3. *Si definisce* procedura di dispiegamento (unfolding) *la procedura che genera, a partire da una rete marcata ordinaria e sana $\langle N, M_0 \rangle$, una rete di occorrenze che si chiamerà appunto dispiegamento di $\langle N, M_0 \rangle$ e che sarà indicata con $\tilde{N}(M_0)$.*

Il dispiegamento descrive in termini strutturali, le possibili evoluzioni della rete di partenza esplicitando tutti i possibili eventi (scatti di transizioni) e le relazioni fra loro.

Tutti i posti ottenuti sono mappati univocamente rispetto alla rete di partenza: a tale scopo si introduce una funzione di etichettatura $\ell : (\tilde{P} \to P) \cup (\tilde{T} \to T)$ che specifica per ogni nodo della rete di dispiegamento qual è il corrispondente nodo della rete originaria. Ad ogni nodo della rete iniziale può invece corrispondere più di un nodo nella rete di dispiegamento. Ciò è dovuto alla proprietà di aciclicità della rete di occorenze. Il fatto che la rete sia aciclica non deve compromettere infatti l'omomorfismo rispetto alla rete di partenza. Il metodo che si usa per mantenere la corrispondenza tra le due reti anche per quel che riguarda i cicli, è quello di replicare i nodi ogni volta che si incontrano. Com'è ovvio tale procedura conduce a una rete che ha un numero di nodi maggiore di quella iniziale.

La procedura di dispiegamento, che sarà ora descritta in dettaglio, è tratta dal lavoro di Giua e Xie (2005). Verrà in seguito fornito un esempio completo di costruzione di una rete di dispiegamento al fine di chiarire tutta la procedura che potrebbe altrimenti risultare troppo astratta o macchinosa.

Procedura 3.1. Dispiegamento*: conversione di una rete sana $\langle N, M_0 \rangle$ in una rete di occorrenze $\tilde{N}(M_0)$.*

1. *Si inizi il dispiegamento inserendo l'insieme dei posti sorgente $\tilde{P}_0$, tale insieme è costituito da tutti i posti che sono marcati nella rete di partenza.*
2. *Si ponga $i := 0$.*
3. *Si ponga $\tilde{P}_{\exp} := \tilde{P}_i$.*
4. *Se $\tilde{P}_i = \emptyset$ STOP.*

5. *Si ponga* $i := i + 1$.

6. $\tilde{P}_i := \emptyset$.

7. *Per ogni transizione* $t \in T$

   Per tutti gli insieme di posti $\tilde{P}' \subseteq \left(\tilde{P}_{\exp} \setminus \tilde{P}_i\right)$ *per i quali siano verificate le seguenti tre condizioni:*

   - $\ell(\tilde{P}') = {}^\bullet t$,
   - *tutti i posti appartenenti a* $\tilde{P}'$ *sono concorrenti,*
   - $\tilde{P}' \cap \tilde{P}_{i-1} \neq \emptyset$,

   (a) Si aggiunga al dispiegamento una nuova transizione $\tilde{t}$ *con etichetta* $\ell(\tilde{t}) = t$.

   (b) Si aggiunga al dispiegamento un nuovo insieme di posti $\tilde{P}''$ *con etichetta* $\ell(\tilde{P}'') = t^\bullet$.

   (c) Si aggiunga un arco da ogni posto appartenente a $\tilde{P}'$ *verso* $\tilde{t}$.

   (d) Si aggiunga un arco da $\tilde{t}$ *verso ogni posto appartenente a* $\tilde{P}''$.

   (e) Si ponga $\tilde{P}_i := \tilde{P}_i \cup \tilde{P}''$.

   (f) Si ponga $\tilde{P}_{\exp} := \tilde{P}_{\exp} \cup \tilde{P}''$.

8. *Si torni al punto 4.*

Si noti che tutte le grandezze segnate col simbolo $\sim$ sono riferite al dispiegamento mentre le altre si riferiscono alla rete originale.

Al primo punto viene richiesto di creare una copia di tutti i posti marcati nella rete di partenza, i quali devono essere messi sulla prima fila che prende il nome di livello 0. In seguito si genera una transizione $\tilde{t}$ per ogni insieme di posti del dispiegamento che soddisfa le tre condizioni elencate al punto sette: la prima è la condizione di abilitazione, cioè i posti del sottoinsieme considerato devono corrispondere a posti che nella rete iniziale abilitavano la transizione t in questione, la seconda condizione impone che i posti siano tutti concorrenti, cioè che possano essere simultaneamente marcati, la terza infine richiede che almeno un posto di questo sottoinsieme faccia parte dell'ultima fila completata, tale condizione serve per garantire che la transizione che è stata generata non sia già stata considerata in uno dei livelli precedenti. Se tutte le condizioni sono soddisfatte, oltre a creare una copia della transizione, si inserisce anche una copia $\tilde{p}$ per ogni posto appartenente all'insieme $t^\bullet$. I posti così ricavati andranno a far parte del successivo livello. E' chiaro come tale procedura possa essere replicata all'infinito, ma saranno introdotte a breve alcune modifiche che permetteranno di fermare l'algoritmo al verificarsi di determinate condizioni. Si sarà notato che non si è ancora parlato di marcature, in realtà è possibile definire una marcatura iniziale anche per la rete di dispiegamento.

Definizione 3.4. *Si definisce marcatura iniziale della rete di dispiegamento* $\tilde{M}_0 = \tilde{P}_0$. *A partire da tale marcatura la rete ha una sua evoluzione e le marcature raggiungibili si indicano con* $\tilde{M} = \left\{ \tilde{p} \in \tilde{P} \,\middle|\, \tilde{M}(\tilde{p}) = 1 \right\}$.

La marcatura iniziale, costituita da tutti i posti contenuti nel livello 0, corrisponde quindi alla marcatura iniziale della rete di origine. Anche tutte le altre marcature $\tilde{M}$ del dispiegamento mantengono un'intrinseca relazione rispetto alle marcature M della rete di partenza. Per esplicitare tale relazione si utilizza la funzione di etichettatura ℓ già introdotta a proposito delle corrispondenze tra i nodi. Ad ogni marcatura $\tilde{M}$ della rete di dispiegamento corrisponde dunque una marcatura della rete di origine $M = \ell(\tilde{M}) \triangleq \left\{ p \in P \,|p = \ell(\tilde{p}) \,, \tilde{p} \in \tilde{M} \right\}$. Questo implica che due marcature $\tilde{M}$ e $\tilde{M}'$ del dispiegamento possano corrispondere alla stessa marcatura M della rete di partenza. Se esiste una relazione di equivalenza di questo tipo, cioè se vale la relazione $\ell(\tilde{M}) = \ell(\tilde{M}')$, allora si scrive $\tilde{M} =_P \tilde{M}'$.

Segue un semplice esempio di applicazione.

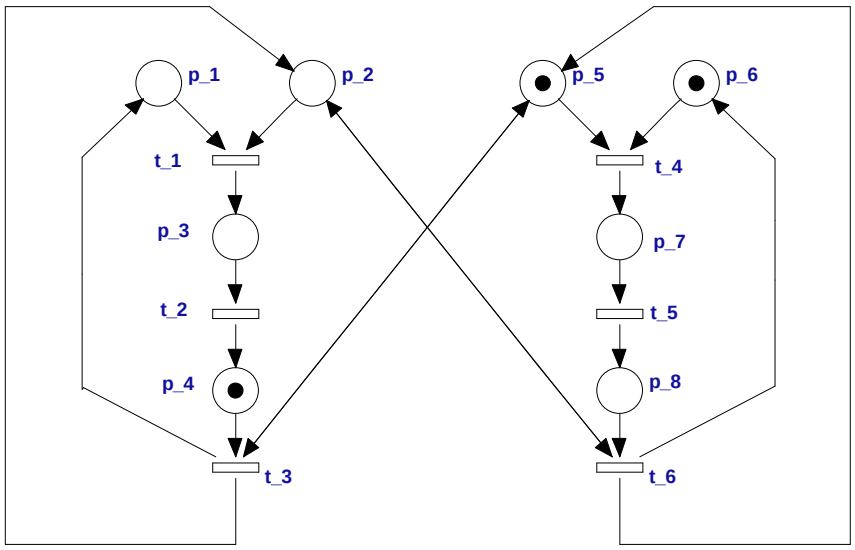

Figura 3.4: Esempio di rete di Petri sana.

Esempio 3.4. Nella rete in Figura 3.4 i posti inizialmente marcati sono p_4, p_5, p_6, pertanto il livello 0 del dispiegamento sarà composto da una copia di tali posti. Le transizioni che possono scattare sono invece t_3 e t_4 e i posti che le abilitano sono rispettivamente $\{p\_4,p\_5\}$ e $\{p\_5, p\_6\}$. Poichè entrambi gli insiemi di posti soddisfano le tre le condizioni richieste, si potrà aggiungere al dispiegamento una copia di t_3 e t_4 e una nuova fila di posti che si chiamerà livello 1 e che conterrà tutti i posti degli insiemi $t\_3^\bullet = \{p\_1, p\_2, p\_5\}$ e $t\_4^\bullet = \{p\_7\}$. Da qui la procedura si ripete; all'aumentare dei livelli è necessario prestare un'attenzione particolare per verificare che i posti che abilitano la transizione siano effettivamente concorrenti. La rete che si ottiene è rappresentata in Figura 3.5. In quest'esempio

il dispiegamento è stato fermato al livello 6 ma dev'essere chiaro che la rete che si ottiene è in realtà di lunghezza infinita.

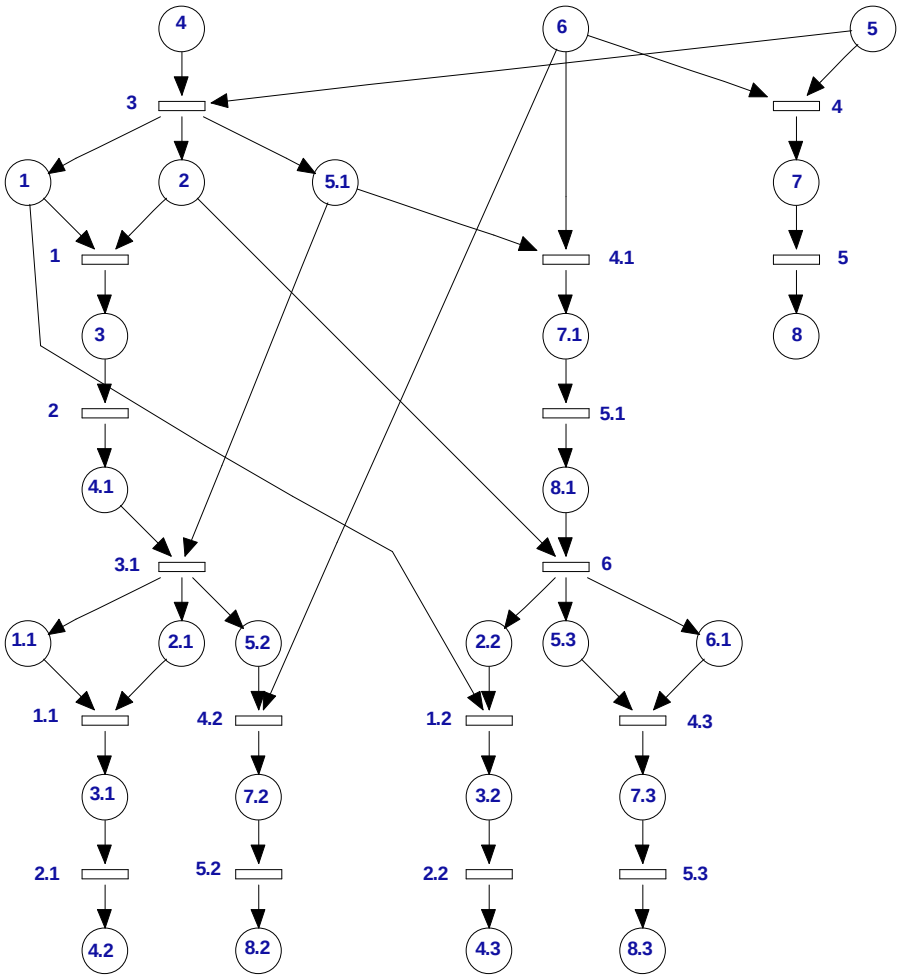

Figura 3.5: Dispiegamento della rete in Figura 3.4

Verranno messe ora in evidenza alcune caratteristiche della rete.

- Per quel che riguarda la concorrenza si può subito notare che i posti $\{p\_4, p\_5.1\}$, pur rispettando la prima e la terza delle tre condizioni, non abilitano la transizione t_3 perchè non sono concorrenti, cioè non possono essere contemporaneamente marcati. I due posti infatti soddisfano la relazione $p\_4 \prec p\_5.1$ e ciò comporta che affichè $p\_5.1$ possa essere marcato p_4 debba perdere il proprio gettone. Lo stesso discorso si può fare per la coppia di posti $\{p\_2.2, p\_1.1\}$ i quali essendo in conflitto (per rendersene conto bisogna risalire al posto p_2) non possono soddisfare la relazione di concorrenza.

- Si noti inoltre il comportamento della rete di dispiegamento in corrispondenza del posto p_8. In quel punto si genera uno stato di blocco che impedisce al sistema di avanzare (le uniche transizioni a poter scattare sarebbero t_3 e t_6 ma non essendo

marcati né p_2 né p_5 questo non è possibile). Nella rete di partenza ciò corrisponde allo scatto delle transizioni t_4, t_5. Potrebbe sembrare incoerente che tale situazione non si riproponga anche per il posto $p\_8.1$. In realtà lo stato di blocco c'è, anche se graficamente non si vede, in quanto il dispiegamento rappresenta in modo simultaneo tutte le evoluzioni possibili, e quindi insieme alla possibilità che successivamente allo scatto di t_3 scattino t_1 e poi t_4 (condizione che porterebbe al blocco) viene contemplata anche la possibilità che scatti solo t_4 e questo permetterebbe, non svuotando il posto p_2 (che funge da risorsa), di evitare il blocco e di replicare così il ciclo.

La rete che è stata ottenuta è una rete di Petri a tutti gli effetti. I posti inizialmente marcati, che com'è ovvio dovranno corrispondere alla marcatura iniziale della rete di partenza, saranno quelli del livello 0 (in quest'esempio i posti p_4, p_5 e p_6). Essi definiranno lo stato iniziale a partire dal quale la rete di dispiegamento avrà una sua propria evoluzione dovuta allo scatto delle transizioni.

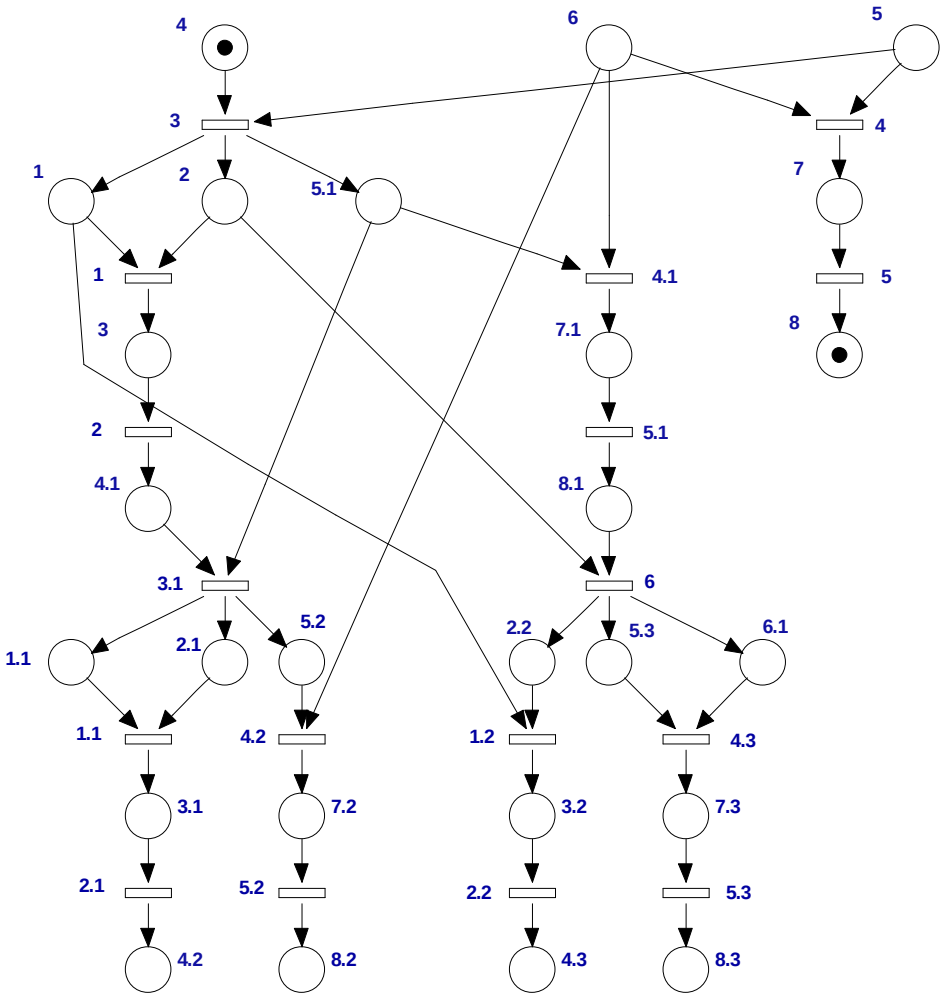

Figura 3.6: Prima marcatura di blocco.

Nel dispiegamento rappresentato in Figura 3.5 lo scatto delle transizioni t_4, t_5 porta a una stato di blocco in cui la marcatura $\tilde{M}$ contiene i posti marcati p_4 e p_8 come mostrato in Figura 3.6. Questo stato non permette al sistema di avanzare, infatti il posto p_5 non è marcato e dunque la transizione t_3 non potrà mai scattare. La condizione di blocco si ripropone anche in Figura 3.7. In tal caso la marcatura $\tilde{M}'$ equivale alla marcatura $\tilde{M}$, cioè è verificata la relazione $\tilde{M} =_P \tilde{M}'$. Per raggiungere la marcatura $\tilde{M}'$ sono scattate le transizioni $t_3, t\_1, t\_4.1, t\_2, t\_5.1$ e a questo punto potrebbero scattare o $t\_3.1$ o t_6, ma nessuna delle due è abilitata in quanto sia il posto p_2 che il posto $p\_5.1$ non sono marcati.

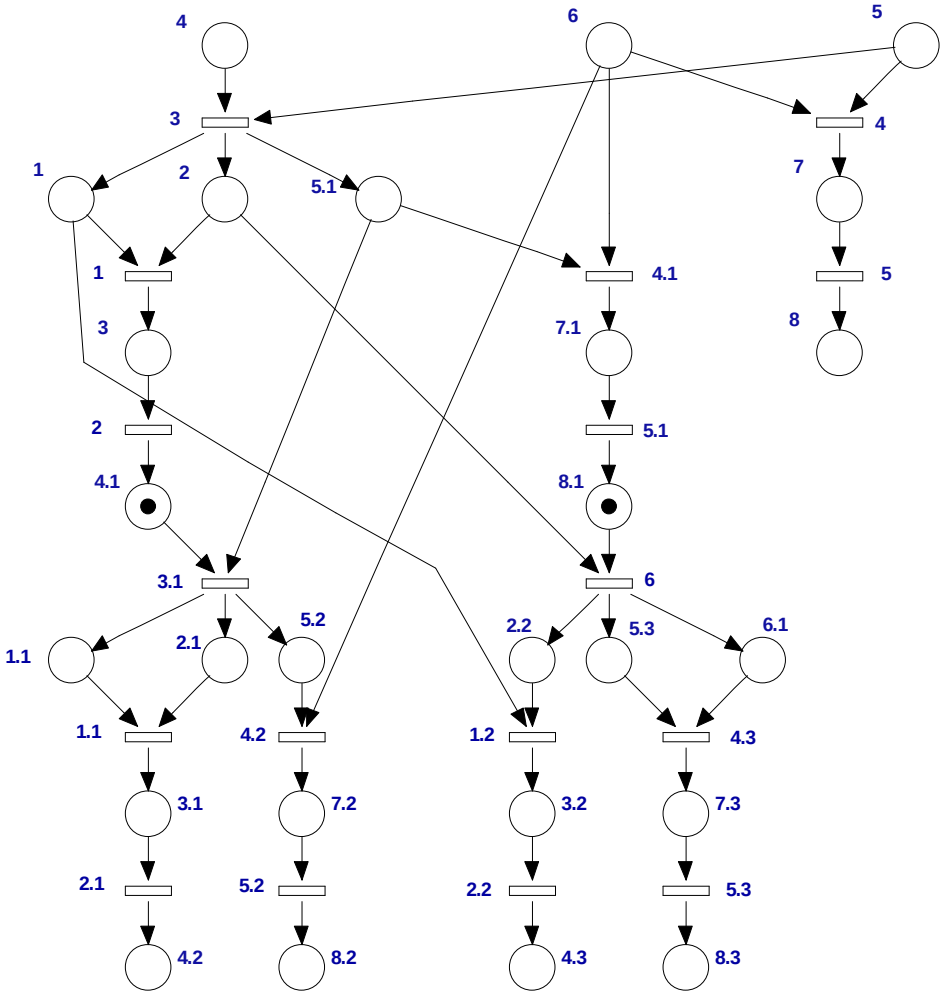

Figura 3.7: Seconda marcatura di blocco.

La raffigurazione della rete di dispiegamento priva di marcatura iniziale non deve quindi farci pensare a queste reti come a un qualcosa di diverso da una rete di Petri vera e propria con un suo stato iniziale e una sua evoluzione. La sua omissione è dovuta unicamente al fatto che la marcatura iniziale è sempre implicitamente rappresentata dai posti contenuti nel livello 0. $\diamond$

3.3 Prefissi finiti

E' stato più volte sottolineato che la procedura descritta in questo modo può proseguire indefinitamente. Le definizioni che saranno introdotte nel seguito hanno lo scopo di permettere la determinazione di alcune condizioni che consentano di tracciare un prefisso finito del dispiegamento, con la garanzia che questo rappresenti comunque tutte le marcature raggiungibili e che da esso si possano trarre tutte le informazioni che in un'analisi tradizionale sono fornite dal grafo di raggiungibilità.

Si cominci con l'enunciare che il vettore di scatto di una rete di occorrenze è un vettore binario poiché ogni transizione può scattare al massimo una volta. Dunque :

Proposizione 3.1. *Si definisce* vettore di scatto *di una rete di occorrenze* $\tilde{X} \in \{0,1\}^{\tilde{n}}$ *un vettore binario che può anche essere visto come un insieme di transizioni* $\tilde{X} = \{\tilde{t} \in \tilde{T} | \tilde{X}(\tilde{t}) = 1\}$.

Dunque in tale vettore si trova un 1 in corrispondenza delle transizioni che sono scattate e 0 altrove.

Esempio 3.5. In Figura 3.8 il vettore di scatto che ha determinato, a partire dalla marcatura iniziale, la marcatura in questione sarà $\tilde{X} = [110101]$. Tale vettore contiene infatti 0 in corrispondenza delle transizioni t_3 e t_5, infatti per ottenere la marcatura in figura queste transizioni non devono scattare, contiene invece un 1 in corrispondenza di t_1, t_2, t_4, t_6 che rappresentano appunto la sequenza di scatto necessaria al raggiungimento dello stato indicato. ◇

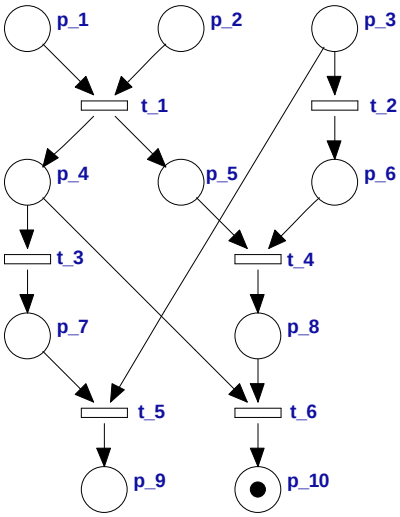

Figura 3.8: Rete di occorrenze

Definizione 3.5. *Data una transizione* $\tilde{t} \in \tilde{T}$ *il minimo vettore di scatto della rete di occor-*

renze che la contiene è chiamato configurazione locale*. Si dimostra che tale vettore è unico e lo si indica con* $[\tilde{t}]$*. La marcatura raggiunta attraverso lo scatto di* $[\tilde{t}]$ *si indica con* $\tilde{M}([\tilde{t}])$*.*

Esempio 3.6. In Figura 3.8 la configurazione locale della transizione t_4 sarà $[t\_4] = [110100]$ questo vettore di scatto è, di tutti quelli che possono essere definiti in tale rete, il più piccolo che contiene t_4. La sequenza di scatto che gli corrisponde è t_1, t_2, t_4, la quale contiene infatti tutte le transizioni in corrispondenza delle quali $[t\_4]$ contiene un 1. $\diamond$

A partire da queste definizioni si possono introdurre vari modelli di prefissi finiti. In particolare verranno date la definizioni di:

- Primo ordine di dispiegamento.
- Secondo ordine di dispiegamento.
- Prefisso finito di McMillan.

3.3.1 Primo ordine di dispiegamento

Definizione 3.6. *Il primo ordine di dispiegamento si indica con* $\tilde{N}_1(M_0)$ *e si ottiene dalla procedura di dispiegamento fermando la costruzione della rete quando si raggiunge una transizione* $\tilde{t}$*, detta di* cut-off *di primo ordine (*cut-off1*), cioè una transizione tale che:*

- *lo scatto della configurazione locale di* $\tilde{t}$ *riporti a una marcatura corrispondente a quella iniziale, cioè tale che* $\tilde{M}([\tilde{t}]) =_P \tilde{M}_0$*;*
- *oppure esista un'altra transizione* $\tilde{t}'$*, detta "specchio" di* $\tilde{t}$*, con le seguenti proprietà:*

  (a) $\tilde{t}'$ *ha una configurazione più piccola di* $\tilde{t}$*:* $[\tilde{t}'] \subset [\tilde{t}]$*;*

  (b) le marcature raggiunte attraverso lo scatto delle due configurazioni sono equivalenti, cioè $\tilde{M}([\tilde{t}']) =_P \tilde{M}([\tilde{t}])$*;*

Per costruire il primo ordine di dispiegamento si utilizza il seguente algoritmo:

Algoritmo 3.1. Primo ordine di dispiegamento*: conversione di una rete sana* $\langle N, M_0 \rangle$ *in una rete di dispiegamento del primo ordine* $\tilde{N}_1(M_0)$*.*

1. *Si inizi il dispiegamento inserendo l'insieme dei posti sorgente* $\tilde{P}_0$*. Tale insieme è costituito da tutti i posti che sono marcati nella rete di partenza.*

2. *Si ponga* $i := 0$.
3. *Si ponga* $\tilde{P}_{\exp} := \tilde{P}_i$.
4. *Se* $\tilde{P}_i = \emptyset$ ***STOP.***
5. *Si ponga* $i := i + 1$.
6. $\tilde{P}_i := \emptyset$.
7. *Per ogni transizione* $t \in T$

   Per tutti gli insieme di posti $\tilde{P}' \subseteq \left(\tilde{P}_{\exp} \setminus \tilde{P}_i\right)$ *per i quali siano verificate le seguenti tre condizioni:*

   - $\ell(\tilde{P}') = {}^\bullet t$,

   - tutti i posti appartenenti a $\tilde{P}'$ *sono concorrenti,*

   - $\tilde{P}' \cap \tilde{P}_{i-1} \neq \emptyset$,

   (a) Si aggiunga al dispiegamento una nuova transizione $\tilde{t}$ *con etichetta* $\ell(\tilde{t}) = t$.

   (b) Si aggiunga al dispiegamento un nuovo insieme di posti $\tilde{P}''$ *con etichetta* $\ell(\tilde{P}'') = t^\bullet$.

   (c) Si aggiunga un arco da ogni posto appartenente a $\tilde{P}'$ *verso* $\tilde{t}$.

   (d) Si aggiunga un arco da $\tilde{t}$ *verso ogni posto appartenente a* $\tilde{P}''$.

   (e) Si ponga $\tilde{P}_i := \tilde{P}_i \cup \tilde{P}''$.

   (f) Se t *non è una transizione di cut-off1 allora si ponga* $\tilde{P}_{\exp} := \tilde{P}_{\exp} \cup \tilde{P}''$.
8. *Si torni al punto 4.*

Rispetto alla Procedura 3.1 è stato modificato il punto *7.(f)*. In questo modo i posti che vengono dopo una transizione di cut-off1 non fanno parte dell'insieme $\tilde{P}_{\exp}$ dal quale si attinge al punto 7 dell'algoritmo per aggiungere nuove transizioni al dispiegamento. Pertanto con tale modifica i posti in uscita da una transizione di cut-off1 non possono mai abilitare nessuna transizione.

Esempio 3.7. Il prefisso finito della Figura 3.9 è stato ottenuto utilizzando questo nuovo algoritmo. Le transizioni segnate con una barra all'interno sono le transizioni di cut-off1. La caratteristica di questa rete è, come spiegato finora, di non avere nessuna transizione abilitata dai posti successivi a una transizione di cut-off1. ◇

In modo perfettamente analogo si può definire un prefisso finito ancora più ampio di quello appena visto: il secondo ordine di dispiegamento.

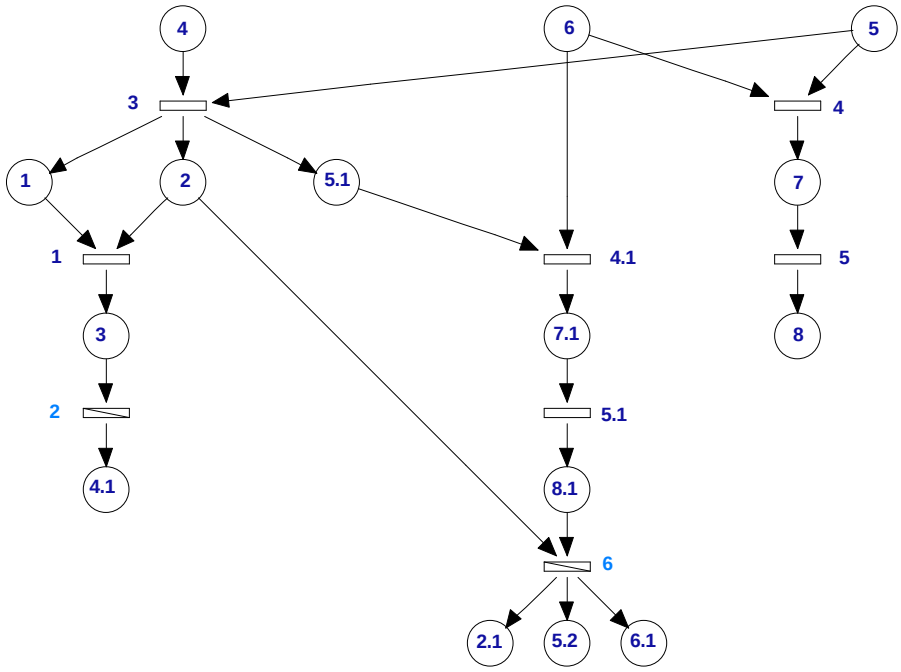

Figura 3.9: Primo ordine di dispiegamento. Le transizioni sbarrate sono quelle di cut-off1.

3.3.2 Secondo ordine di dispiegamento

Definizione 3.7. *Una volta costruito $\tilde{N}_1(M_0)$, si supponga di voler continuare il dispiegamento fino a quando non si raggiunge una transizione $\tilde{t}$, detta transizione di* cut-off *di secondo ordine (*cut-off2*), tale che esista una transizione $\tilde{t}'$ con le seguenti proprietà:*

(a) $\tilde{t}'$ non appartiene a $\tilde{N}_1(M_0)$ o al massimo è una transizione di cut-off1;

(b) $\tilde{t}'$ ha una configurazione più piccola di $\tilde{t}$:$[\tilde{t}'] \subset [\tilde{t}]$;

(c) le due configurazioni raggiungono marcature che sono equivalenti;

La rete così ricavata rappresenta il secondo ordine di dispiegamento, e si indica con $\tilde{N}_2(M_0)$.

Le considerazioni già fatte per il primo ordine si possono ripetere in maniera analoga per il secondo. Anche in questo caso è necessario modificare il punto *7.(f)* della procedura 3.1 in modo tale da impedire che una transizione venga abilitata da posti dell'insieme $\tilde{t}^\bullet$ quando $\tilde{t}$ è una transizione di cut-off2. L'algoritmo diventa:

Algoritmo 3.2. Secondo ordine di dispiegamento*: conversione di una rete sana $\langle N, M_0 \rangle$ in una rete di dispiegamento del secondo ordine $\tilde{N}_2(M_0)$.*

1. *Si inizi il dispiegamento inserendo l'insieme dei posti sorgente $\tilde{P}_0$, il quale è costituito da tutti i posti che sono marcati nella rete di partenza.*

2. *Si ponga* $i := 0$.
3. *Si ponga* $\tilde{P}_{\exp} := \tilde{P}_i$.
4. *Se* $\tilde{P}_i = \emptyset$ *STOP.*
5. *Si ponga* $i := i + 1$.
6. $\tilde{P}_i := \emptyset$.
7. *Per ogni transizione* $t \in T$

   Per tutti gli insieme di posti $\tilde{P}' \subseteq \left(\tilde{P}_{\exp} \setminus \tilde{P}_i\right)$ *per i quali siano verificate le seguenti tre condizioni:*

   - $\ell(\tilde{P}') = {}^\bullet t$,

   - tutti i posti appartenenti a $\tilde{P}'$ *sono concorrenti,*

   - $\tilde{P}' \cap \tilde{P}_{i-1} \neq \emptyset$,

   (a) *Si aggiunga al dispiegamento una nuova transizione* $\tilde{t}$ *con etichetta* $\ell(\tilde{t}) = t$.

   (b) *Si aggiunga al dispiegamento un nuovo insieme di posti* $\tilde{P}''$ *con etichetta* $\ell(\tilde{P}'') = t^\bullet$.

   (c) *Si aggiunga un arco da ogni posto appartenente a* $\tilde{P}'$ *verso* $\tilde{t}$.

   (d) *Si aggiunga un arco da* $\tilde{t}$ *verso ogni posto appartenente a* $\tilde{P}''$.

   (e) *Si ponga* $\tilde{P}_i := \tilde{P}_i \cup \tilde{P}''$.

   (f) *Se* t *non è una transizione di cut-off2 allora si ponga* $\tilde{P}_{\exp} := \tilde{P}_{\exp} \cup \tilde{P}''$.
8. *Si torni al punto 4.*

Esempio 3.8. La rete in Figura 3.10 rappresenta il secondo ordine di dispiegamento della rete in Figura 3.4. Come si vede il dispiegamento continua dopo che sono state raggiunte le transizioni di cut-off1 finché non si incontra una transizione che verifica le condizioni della Definizione 3.8. ◇

La procedura può esser estesa anche a casi più generali e si può così definire in modo perfettamente analogo un dispiegamento di ordine qualsiasi.

3.3.3 Prefisso finito di McMillan

McMillan fu il primo a presentare, nel 1995, una nuova tecnica per costruire un prefisso finito di dispiegamento. Essa costituirà un'importante fonte di studio per tutti i ricercatori sull'argomento in questione.

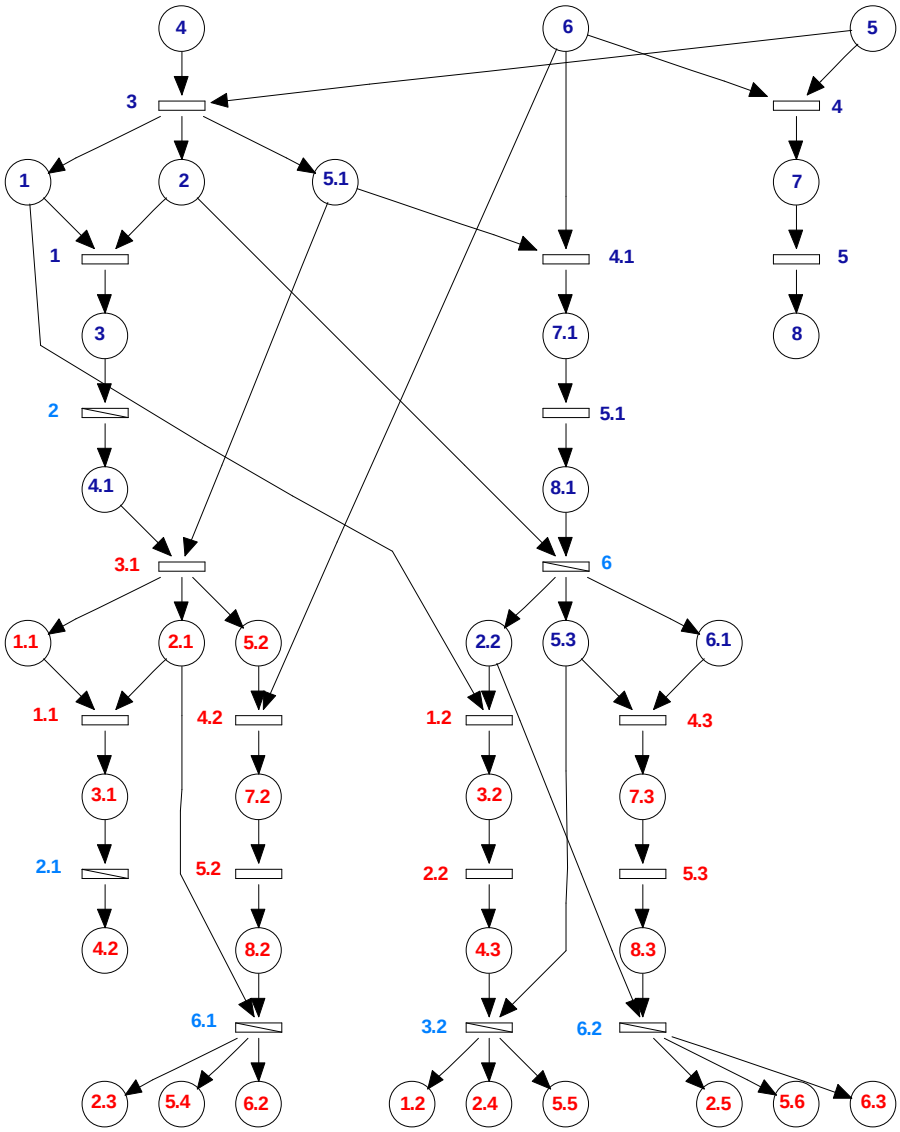

Figura 3.10: Secondo ordine di dispiegamento della rete in Figura 3.4.

Definizione 3.8. *La costruzione della rete di dispiegamento viene fermata quando si raggiunge una transizione* $\tilde{t}$ *di* cut-off *secondo McMillan, cioè una transizione tale che:*

- *lo scatto della configurazione locale di* $\tilde{t}$ *riporti alla marcatura iniziale*
- *oppure esista un'altra transizione* $\tilde{t}'$ *con le seguenti proprietà:*

  (a) $\tilde{t}'$ *ha una configurazione più piccola di* $\tilde{t}$*:* $card([\tilde{t}']) < card([\tilde{t}])$*;*

  (b) le marcature raggiunte attraverso lo scatto delle due configurazioni sono equivalenti;

L'algoritmo completo che consente di ottenere tale prefisso finito sarà quindi:

Algoritmo 3.3. Prefisso finito di McMillan*: conversione di una rete sana* $\langle N, M_0 \rangle$ *in un prefisso finito di McMillan* $\tilde{N}_{McM}(M_0)$*.*

1. *Si inizi il dispiegamento inserendo l'insieme dei posti sorgente* $\tilde{P}_0$*, il quale è costituito da tutti i posti che sono marcati nella rete di partenza.*
2. *Si ponga* $i := 0$*.*
3. *Si ponga* $\tilde{P}_{\text{exp}} := \tilde{P}_i$*.*
4. *Se* $\tilde{P}_i = \emptyset$ ***STOP.***
5. *Si ponga* $i := i + 1$*.*
6. $\tilde{P}_i := \emptyset$*.*
7. *Per ogni transizione* $t \in T$

   Per tutti gli insieme di posti $\tilde{P}' \subseteq \left(\tilde{P}_{\text{exp}} \setminus \tilde{P}_i\right)$ *per i quali siano verificate le seguenti tre condizioni:*

   - $\ell(\tilde{P}') = {}^\bullet t$,
   - *tutti i posti appartenenti a* $\tilde{P}'$ *sono concorrenti,*
   - $\tilde{P}' \cap \tilde{P}_{i-1} \neq \emptyset$,

   (a) Si aggiunga al dispiegamento una nuova transizione $\tilde{t}$ *con etichetta* $\ell(\tilde{t}) = t$*.*

   (b) Si aggiunga al dispiegamento un nuovo insieme di posti $\tilde{P}''$ *con etichetta* $\ell(\tilde{P}'') = t^\bullet$*.*

   (c) Si aggiunga un arco da ogni posto appartenente a $\tilde{P}'$ *verso* $\tilde{t}$*.*

   (d) Si aggiunga un arco da $\tilde{t}$ *verso ogni posto appartenente a* $\tilde{P}''$*.*

   (e) Si ponga $\tilde{P}_i := \tilde{P}_i \cup \tilde{P}''$*.*

   (f) Se t *non è una transizione di cut-off secondo McMillan allora si ponga* $\tilde{P}_{\text{exp}} := \tilde{P}_{\text{exp}} \cup \tilde{P}''$*.*
8. *Si torni al punto 4.*

Come si può ben notare, il prefisso finito di McMillan è assai simile al primo ordine di dispiegamento. L'unica differenza consiste nel fatto che, mentre nel primo ordine la relazione tra la configurazione locale della transizione di cut-off e la configurazione locale della *transizione specchio* deve essere di inclusione ($[\tilde{t}'] \subset [\tilde{t}]$) , nel prefisso finito di McMillan tale condizione è meno forte e prevede unicamente che card($[\tilde{t}']$) $<$ card($[\tilde{t}]$). Questo conduce in generale a un prefisso ancora più breve del primo ordine di dispiegamento in quanto la transizione specchio può trovarsi anche in un ramo del dispiegamento che non è quello della transizione di cut-off.

Esempio 3.9. Per chiarire meglio quest'ultimo concetto, vengono forniti il primo ordine di dispiegamento (Figura 3.12) e il prefisso finito di McMillan (Figura 3.13) della rete in Figura 3.11 (Benveniste, 2003). La differenza tra i due prefissi finiti si trova nella transizione t_6. Nel prefisso finito di McMillan, infatti, questa è una transizione di cut-off mentre non è così nel primo ordine di dispiegamento.

Affinchè la transizione t_6 possa essere considerata una transizione di cut-off del primo ordine di dispiegamento il suo scatto dovrebbe riportare a una marcatura M equivalente a quella iniziale $M0 = [1001001]$ oppure dovrebbe esistere una transizione specchio, cioè una transizione che appartenga alla sua sequenza di scatto, che conduca a una marcatura equivalente ad M stessa. La sequenza di scatto di t_6 sarà t_1, t_4, t_6 e la marcatura a cui essa conduce equivale nella rete di partenza (quella di Figura 3.11) a $M = [0101001]$. Tale marcatura non è certamente quella iniziale, e non corrisponde neanche alle marcature che si raggiungono tramite lo scatto di t_1 ($M1 = [0111000]$) o di t_1, t_4 ($M2 = [0100101]$) . E' per questo motivo che t_6 non sarà certamente una transizione di cut-off del primo ordine di dispiegamento.

Le cose cambiano se si considera la definizione di transizione di cut-off data da McMillan. In tal caso infatti non è necessario che la transizione specchio si trovi nella sequenza di scatto di t_6 ma è sufficiente che abbia una sequenza di scatto più breve e che conduca a una marcatura equivalente ad M. In Figura 3.13 si può vedere che lo scatto di t_2 conduce a una marcatura che nella rete di partenza equivale a $M3 = [0101001]$. Dunque, poichè $M3 = M$, t_6 è una transizione di cut-off secondo McMillan. ◇

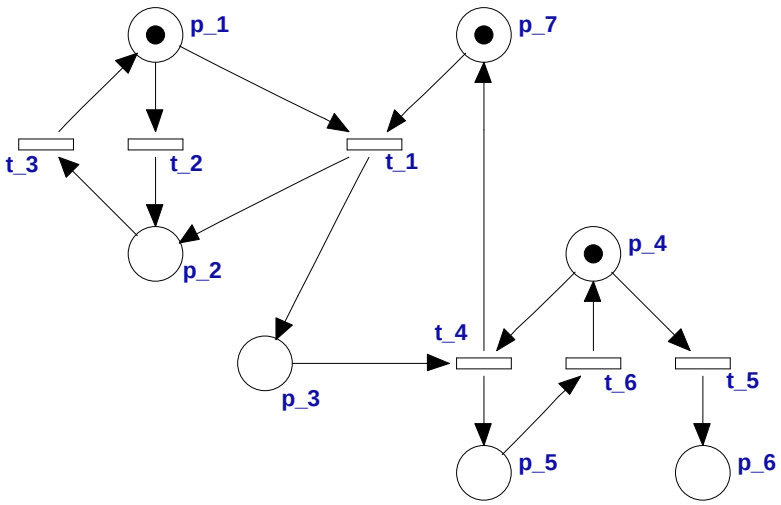

Figura 3.11: Rete posto/transizione

Il prefisso finito di McMillan consente di rappresentare tutte le marcature dell'insieme di raggiungibilità della rete originale. Tale risultato è stato dimostrato da McMillan (1995).

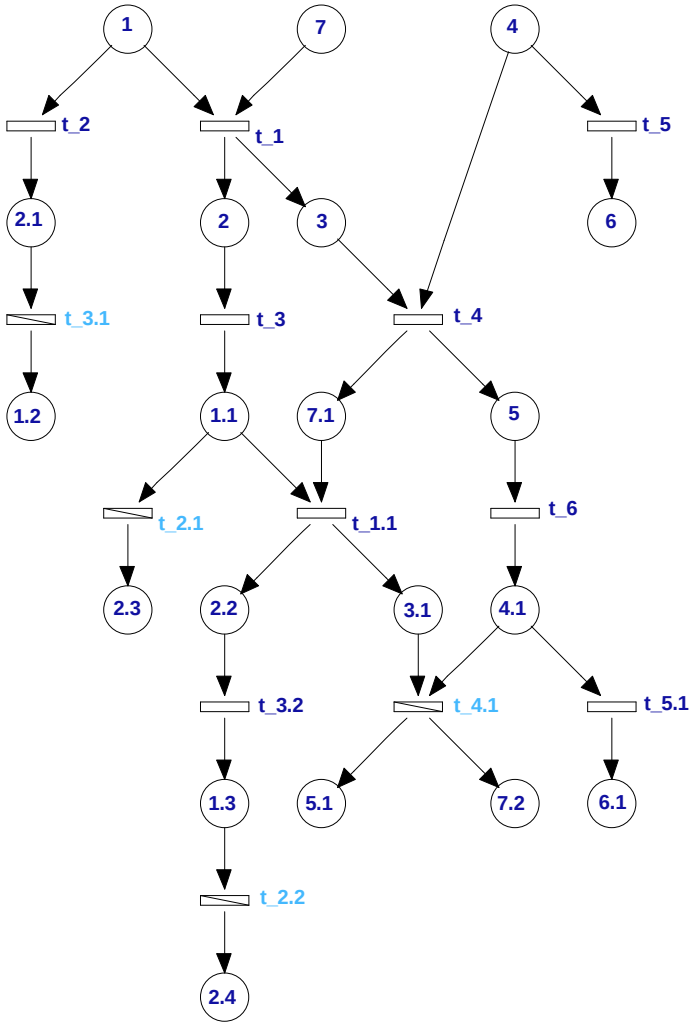

Figura 3.12: Primo ordine di dispiegamento della rete in Figura 3.11.

Se questo è vero per il prefisso di McMillan, lo sarà a maggior ragione anche per il primo e il secondo ordine di dispiegamento che sono dei prefissi più lunghi.

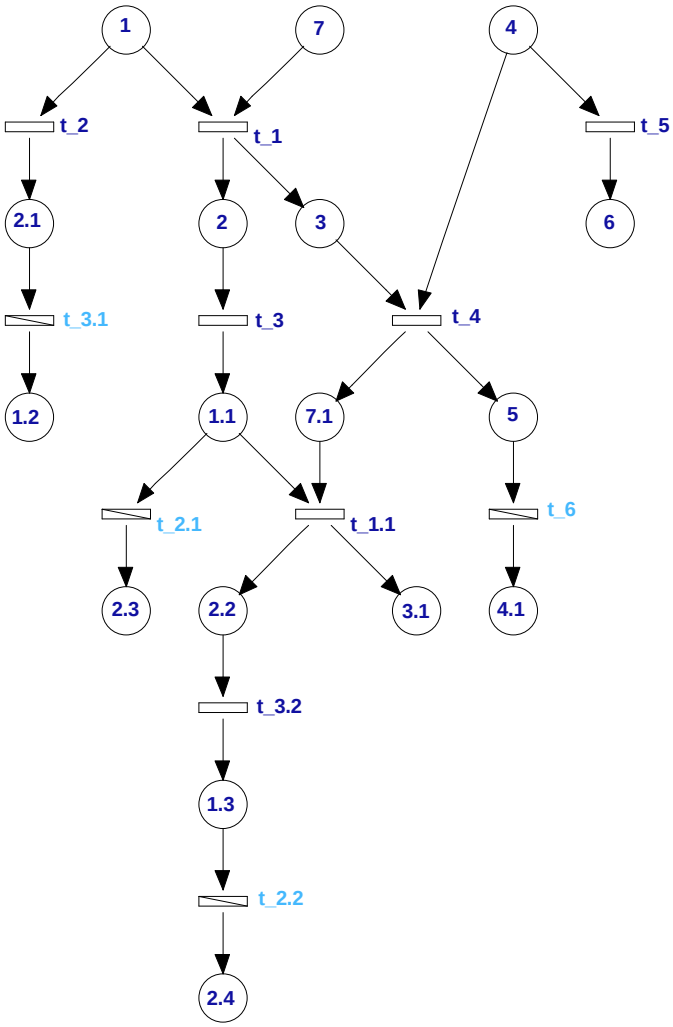

Figura 3.13: Prefisso finito di McMillan della rete in Figura 3.11.

Capitolo 4

Principali funzioni del toolbox Matlab

In questo capitolo verranno illustrate le funzioni che compongono il pacchetto Matlab. Innanzitutto si descriverà l'implementazione della procedura generale di dispiegamento per poi entrare più nello specifico e distinguere i diversi casi a seconda del prefisso finito che si intende utilizzare. Infine verranno fornite alcune *utilities* volte a facilitare l'analisi delle reti di Petri mediante l'utilizzo del dispiegamento.

4.1 Funzione unfolding_n

E' la più generica delle funzioni perché non si riferisce ad un prefisso finito in particolare. Essa implementa in maniera abbastanza fedele la procedura presentata nel capitolo precedente. Si supponga di avere una rete di Petri sana e ordinaria $\langle N, M_0 \rangle$ e di voler determinare il suo dispiegamento $\tilde{N}(M_0)$. Non verrà specificata per ora la scelta di un particolare prefisso finito; si costruirà un dispiegamento fermandolo ad un desiderato livello di profondità (o numero di livelli).

4.1.1 Sintassi

L'interfaccia della funzione è la seguente:

```
[Pre_u,Post_u,p_tier,t_tier,Pexp,Texp]=unfolding_n(Pre,Post,M0,f)
```

I parametri in ingresso sono dunque:

- `Pre` : la matrice di pre-incidenza della rete $\langle N, M_0 \rangle$.

- `Post`: la matrice di post-incidenza della rete $\langle N, M_0 \rangle$.
- `M0`: la marcatura iniziale della rete $\langle N, M_0 \rangle$.
- `f`: livello desiderato di profondità della rete di dispiegamento.

Le uscite saranno invece:

- `Pre_u`: la matrice di pre-incidenza della rete $\tilde{N}(M_0)$.
- `Post_u`: la matrice di post-incidenza dell rete $\tilde{N}(M_0)$.
- `p_tier`: quest'uscita è strutturata come un array di celle: nella *i-esima* cella sono indicate le etichette dei posti del *(i - 1)-esimo* livello della rete di dispiegamento che rendono conto della corrispondenza con i posti appartenenti alla rete iniziale (si noti che gli indici in Matlab partono da uno mentre in questo lavoro la prima fila di posti è stata convenzionalmente indicata come livello 0).
- `t_tier`: anche quest'uscita è strutturata come un array di celle: nella *i-esima* cella sono indicate le etichette delle transizioni del *(i - 1)-esimo* livello della rete di dispiegamento che rendono conto della corrispondenza con le transizioni appartenenti alla rete iniziale.
- `Pexp`: il suo utilizzo è assai simile al Pexp presentato nella procedura proposta nel Capitolo 3. E' un vettore al cui interno si trovano tutti i posti della rete di dispiegamento ognuno etichettato con il nome del corrispondente posto della rete di partenza.
- `Texp`: Analogamente al `Pexp` si tratta di un vettore al cui interno si trovano tutte le transizioni della rete di dispiegamento ognuna etichettata con il nome della corrispondente transizione della rete di partenza.

Ad ogni nodo della rete di dispiegamento sono associate due numerazioni: la prima considera i posti del dispiegamento in ordine crescente come se questi appartenessero a una rete di Petri indipendente e sono quelli che consentono di generare le matrici `Pre_u` e `Post_u`; la seconda, definita tramite le uscite `Pexp` e `Texp`, tiene invece conto della numerazione originale ed assolve alla funzione di etichettatura $\ell : (\tilde{P} \rightarrow P) \cup (\tilde{T} \rightarrow T)$ descritta nel precedente capitolo. La corrispondenza tra le due è sempre definita. Gli indici dei vettori `Pexp`, `Texp` rappresentano infatti la numerazione del dispiegamento mentre il contenuto delle rispettive celle mostra a quale nodo della rete originale corrisponde quel nodo della rete di dispiegamento. Per evitare l'ambiguità, causata dal fatto che a più posti della rete di dispiegamento corrisponde lo stesso posto della rete originale, è stata introdotta in tutte le rappresentazioni grafiche una particolare numerazione per distinguere i vari posti che avrebbero altrimenti la stessa etichetta . Viene fornito un esempio per chiarire la logica di tale numerazione.

Esempio 4.1. Nella rete di dispiegamento in Figura 3.10 i posti 5, 5.1, 5.2, 5.3, 5.4 etc. corrisponderanno allo stesso posto p_5 della rete di partenza rappresentata in figura 3.4 (questa numerazione vale per tutte le reti di dispiegamento). $\diamond$

4.1.2 Descrizione del programma

Sarà ora descritta in dettaglio la funzione. Essa si apre con una serie di definizioni di insiemi di base e di inizializzazioni di vettori i cui contenuti verranno integrati nell'evolvere della procedura. Si definiscono quindi:

1. Un insieme `P0` che contiene i posti inizialmente marcati: nella rete di dispiegamento questi andranno a formare il livello 0.
2. Un insieme `P0ord` che rinumera i posti della prima fila in ordine crescente.
3. Un vettore `Pexp` che contiene i posti trovati e al quale verranno man mano aggiunti tutti i posti del dispiegamento.
4. Un insieme `Pexp_ord` che rinumera tutti i posti di `Pexp` in ordine crescente.
5. Una matrice `mat_conc` che indica per ogni posto della rete di dispiegamento quali sono i posti ad esso concorrenti.

Dunque fino ad ora è nota solo la prima fila di posti del dispiegamento; è già possibile tuttavia inizializzare gli insiemi di uscita.

```
Pexp=P0; p_tier(1)={P0}.
```

Si è già visto come di alcuni insiemi di posti ci siano due versioni (`P0` e `P0ord`; `Pexp` e `Pexp_ord`): esse sono relative alle due numerazioni usate. In particolare in tutti gli insiemi in cui si incontra il suffisso `_ord` si intende indicare la numerazione in ordine crescente, mentre ogni volta che tale suffiso non è presente si vuole indicare l'insieme dei posti con la numerazione della rete di partenza.

A questo punto inizia la procedura vera e propria. Per prima cosa si crea un ciclo

```
for g=1:f
```

che consente di ripetere i passaggi dell'algoritmo `f` volte (dove `f` è il parametro di ingresso che indica il numero di livelli di dispiegamento desiderati). Per ogni transizione $\tilde{t}$ della rete $\langle N, M_0 \rangle$ si eseguono i seguenti punti:

1. Si crea un vettore `Ptot_ord` che contiene l'insieme dei posti della rete $\tilde{N}(M_0)$ ad esclusione di quelli trovati nel livello corrente.

2. Si costruisce il corrispondente vettore `Ptot` che contiene lo stesso insieme di posti ma numerati come nella rete $\langle N, M_0 \rangle$.

3. Si definisce un vettore `ind`, che contiene i posti della rete $\langle N, M_0 \rangle$ che abilitano la transizione corrente.

4. Si costruisce una matrice `Psubtot` ottenuta mettendo insieme opportunamente i vettori `Ptot` e `ind`: nelle righe sono contenute tutte le combinazioni di posti della rete di dispiegamento che corrispondono a posti della rete iniziale che abilitano la transizione. Nella Procedura descritta nel capitolo 3 ciò corrisponde alla prima condizione del punto *7*.

5. Per ogni riga di `Psubtot`, che verrà indicata con `h`, devono essere verificate anche le altre due condizioni; solo in tal caso infatti la transizione corrente sarà una transizione del dispiegamento.

   Viene controllata prima la terza condizione; a tale scopo si introduce il vettore `P_old_ord` che contiene i posti dell'ultimo livello completato in precedenza. Affinché la terza condizione sia verificata è necessario che l'intersezione tra il vettore `P_old_ord` e la riga di `Psubtot` che si sta processando non sia un insieme vuoto. Se questo è vero allora si può passare alla verifica della seconda condizione. Questa impone che i posti appartenenti ad `h` siano tutti concorrenti.

   In un primo momento era stata proposta una soluzione che consisteva nel determinare la concorrenza attraverso la definizione descritta al Capitolo 3. Tale idea, anche se corretta, risultava troppo complessa dal punto di vista computazionale e quindi molto lenta nel fornire le uscite. Si è intrapresa quindi un'altra strada che si avvale della matrice di concorrenza `mat_conc` . In un paragrafo successivo verranno messe a confronto le due soluzioni e verrà così dimostrata la maggiore efficienza della seconda scelta adottata.

   La matrice di concorrenza è una matrice simmetrica; ad ogni riga e ad ogni colonna è associato un posto del dispiegamento (la numerazione considerata è ovviamente quella crescente) gli elementi della matrice possono essere solo 1 oppure 0: si trova 1 in corrispondenza dei posti che sono tra loro concorrenti e 0 in corrispondenza dei posti che non lo sono. E' chiaro come nella diagonale principale di una matrice così definita non possano esserci che zeri in quanto un posto non può per definizione mai essere concorrente con se stesso .

   I posti contenuti nel vettore `h` devono essere tutti concorrenti tra loro dunque, la sottomatrice, che si estrae da mat_conc considerando solo le righe e le colonne dei posti

di `h`, deve essere una matrice che abbia 1 dappertutto eccetto che nella diagonale principale. Se tale condizione è verificata allora i posti sono concorrenti e la transizione $\tilde{t}$ può effettivamente dirsi appartenente al grafo di dispiegamento.

6. Si possono ora eseguire tutti i passaggi da *(a)* a *(f)* indicati al punto 7 dell'algoritmo:

   (a) Si aggiunge al dispiegamento una nuova transizione: ciò equivale ad aggiungere tale transizione alle uscite `Texp` e `t_tier`. Si crea inoltre una nuova colonna, corrispondente alla transizione corrente, in `Pre_u` e in `Post_u`.

   (b) Si crea un vettore `P11` contenente l'insieme dei posti $\tilde{t}^{\bullet}$.

   (c) (d) Richiedono di aggiungere gli archi pre e post al dispiegamento per la transizione corrente. In questa funzione questa specifica grafica corrisponde alla compilazione delle matrici `Pre_u` e `Post_u`.

   (e) Utilizzando l'insieme `P11` viene poi aggiornato il vettore `P1` che rappresenta tutti i posti del livello corrente.

   (f) Sempre con il vettore `P11` si aggiorna l'insieme `Pexp`.

A questo punto non resta che aggiornare le variabili `p_tier`, `t_tier` e `mat_conc` e si ripete tutto il ciclo.

Esempio 4.2. La rete posto/transizione in Figura 4.1 verrà utilizzata per mostrare come si presentano gli ingressi e le uscite nell'interfaccia Matlab.

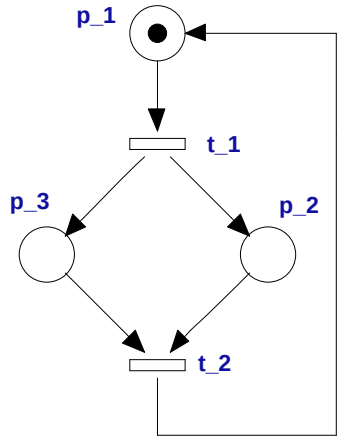

Figura 4.1: Rete posto/transizione

Si ricorda l'interfaccia della funzione:

```
[Pre_u,Post_u,p_tier,t_tier,Pexp,Texp]=unfolding_n(Pre,Post,M0,f)
```

I dati da fornire in ingresso saranno:

```
Pre=[1 0; 0 1; 0 1];
Post=[0 1;1 0;1 0];
M0=[1 0 0]';
f=3;
```

Con `f=3` la procedura di dispiegamento viene fermata al livello 3 (si ricordi che poichè la prima fila di posti corrisponde al livello 0, il livello 3 corrisponderà alla quarta fila di posti). I risultati forniti in uscita da Matlab saranno:

```
Pre_u =

     1     0     0
     0     1     0
     0     1     0
     0     0     1
     0     0     0
     0     0     0

Post_u =

     0     0     0
     1     0     0
     1     0     0
     0     1     0
     0     0     1
     0     0     1

p_tier =

    [1]    [1x2 double]    [1]    [1x2 double]

t_tier =

    [1]    [2]    [1]

Pexp =

     1     2     3     1     2     3

Texp =

     1     2     1
```

Per visualizzare il contenuto dell'array di celle `p_tier` si utilizza il comando `celldisp` di Matlab :

```
p_tier{1} =

     1

p_tier{2} =

     2     3

p_tier{3} =

     1

p_tier{4} =

     2     3
```

Graficamente ciò che si ottiene è rappresentato in Figura 4.2. ◇

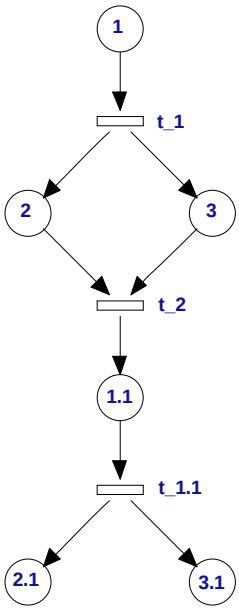

Figura 4.2: Livello 3 di dispiegamento della rete in Figura 4.1

4.1.3 Procedure per la valutazione della concorrenza

E' già stato accennato in precedenza il duplice approccio utilizzato nell'analisi della concorrenza. La prima strada intrapresa utilizzava la funzione `is_concurrent`. Sarà analizzata ora l'interfaccia:

```
[pp]=isconcurrent(Pre_u,Post_u,p1,t_tier)
```

Tra gli ingressi l'unica variabile di cui ancora non si è parlato è `p1`. Esso è un vettore che contiene l'insieme dei posti di cui si vuole verificare la concorrenza. La funzione restituisce `pp=1` se i posti sono concorrenti e `pp=0` altrimenti.

Tale funzione si basa sulla definizione di concorrenza fornita nel Capitolo 3, secondo cui due nodi x e x' si dicono concorrenti se nessuna delle seguenti relazioni è verificata:

1. $x \preceq x'$,
2. $x' \preceq x$,
3. $x \# x'$.

Ad ognuna di queste relazioni è stata associata una funzione. In particolare le prime due condizioni vengono verificate dalla funzione:

```
[posti3]=previous(Pre,Post,posti,t_tier)
```

mentre la terza dalla funzione:

```
[p]=conflict(Pre,Post,Px,t_tier)
```

La funzione `previous` restituisce in uscita tutti i posti `posti3` che vengono prima del vettore `posti` passato in ingresso.

Invece la funzione `conflict` prende in ingresso un generico posto `Px` e restitusce tutti i posti `p` del dispiegamento che sono in conflitto con esso.

La funzione `is_concurrent` si serve di queste due funzioni per determinare se i posti in ingresso sono concorrenti o meno.

Questo metodo aveva fornito dei risultati non soddisfacenti in termini di efficienza computazionale. Per questa ragione si è scelto di adottare una soluzione alternativa. Ci si è dunque avvalsi di uno strumento costruito ad hoc già introdotto col nome di matrice di concorrenza. Verranno ora spiegate le modalità di costruzione di quest'ultima attraverso un esempio.

Esempio 4.3. Si consideri la rete in Figura 3.4 e si costruisca il suo dispiegamento fermato al livello 3. Inizialmente i posti marcati sono p_4, p_5, p_6 ed essi sono ovviamente concorrenti. La struttura che prende il nome di `mat_conc` verrà quindi inizializzata come:

```
mat_conc =

          p4  p5  p6

    p4    0   1   1
    p5    1   0   1
    p6    1   1   0
```

Come si può osservare la matrice è simmetrica, in quanto alle righe e alle colonne che hanno lo stesso indice è associato uno stesso posto del dispiegamento. In questo particolare caso si tratta dei posti del livello 0. A questo punto le transizioni abilitate sono due: t_3, t_4. Tali transizioni generano rispettivamente i posti p_1, p_2, $p\_5.1$ e p_7, i quali andranno ad aggiornare la matrice di concorrenza con quattro nuove righe e colonne. Essa verrà quindi aggiornata ogni volta che si completa un nuovo livello e sarà sempre costruita attraverso la concatenazione di quattro sottomatrici secondo lo schema:

$$\left[\begin{array}{c|c} \texttt{mat\_conc1} & \texttt{mat\_conc3} \\ \hline \texttt{mat\_conc2} & \texttt{mat\_conc4} \end{array}\right]$$

Ognuna di esse si ottiene attraverso la seguente procedura:

- **mat_conc1**: è sempre uguale alla `mat_conc` che contiene tutti i posti appartenenti a livelli che precedono quello corrente. Nella iterazione successiva dell'esempio considerato sarà quindi:

```
mat_conc1 =

          p4  p5  p6

    p4    0   1   1
    p5    1   0   1
    p6    1   1   0
```

- **mat_conc2**: determina la concorrenza tra i posti del livello corrente e quelli dei vecchi livelli. Per ottenere la riga corrispondente al posto p_x e sufficiente fare un *and* logico tra le righe di `mat_conc1` che corrispondono a posti "padri" di p_x (un qualsiasi posto potrà infatti essere concorrente di p_x solo se era concorrente di tutti i suoi "padri"). Nel caso in questione i nuovi posti sono: $\{p\_1, p\_2, p\_5.1, p\_7\}$ per ottenere le righe corrispondenti a p_1, p_2 e $p\_5.1$ sarà quindi sufficiente fare un *and* bit a bit tra le righe p_4 e p_5 di `mat_conc1`, mentre per quanto riguarda la riga corrispondente a p_7 si dovrà fare un *and* tra le righe p_5 e p_6 di `mat_conc1`. La matrice che si ottiene è:

```
mat_conc2 =

           p4  p5  p6

     p1    0   0   1
     p2    0   0   1
     p5.1  0   0   1
     p.7   1   0   0
```

- **mat_conc3**: è molto semplicemente la trasposta di `mat_conc2`, nell'esempio si ottiene:

```
mat_conc3 =

           p1  p2 p5.1 p7

     p4    0   0   0   1
     p5    0   0   0   0
     p6    1   1   1   0
```

- **mat_conc4**: in questa matrice sono invece contenute le informazioni relative alla concorrenza tra posti dello stesso livello. Il suo completamento è stato effettuato in due parti; in un primo momento per calcolare ogni riga di `mat_conc4` relativa a un dato posto p_x, è stato fatto l'*and* tra le righe di `mat_conc3` corrispondenti ai posti "padri" di p_x in modo del tutto analogo a quanto visto nella `mat_conc2`. Nel caso in cui ci siano più posti generati dalla stessa transizione $\tilde{t}$ (come nel caso appena visto di$\{p_1, p_2, p_5.1\}$ tutti quanti generati da t_4) la procedura ha bisogno anche di una seconda fase nella quale vengono sovrascritte in corrispondenza di tali posti

delle sottomatrici di dimensioni pari alla cardinalità dell'insieme $\tilde{t}^{\bullet}$ contenenti 0 nella diagonale principale e 1 altrove; infatti se un insieme di posti proviene da una stessa transizione per forza di cose sarà costituito da posti concorrenti.

(a) prima fase:

```
mat_conc4 =

           p1  p2 p5.1 p7

    p1     0   0   0   0
    p2     0   0   0   0
    p5.1   0   0   0   0
    p7     0   0   0   0
```

(b) seconda fase:

```
mat_conc4 =

           p1  p2 p5.1 p7

    p1     0   1   1   0
    p2     1   0   1   0
    p5.1   1   1   0   0
    p7     0   0   0   0
```

La concatenazione delle 4 sottomatrici fornisce la `mat_conc` completa:

```
mat_conc =

          p4  p5  p6  p1  p2 p5.1 p7

    p4     0   1   1   0   0   0   1
    p5     1   0   1   0   0   0   0
    p6     1   1   0   1   1   1   0
    p1     0   0   1   0   1   1   0
    p2     0   0   1   1   0   1   0
    p5.1   0   0   1   1   1   0   0
    p7     1   0   0   0   0   0   0
```

Continuando tale procedura fino al livello 3 si ottiene la seguente matrice:

```
mat_conc =

        p4  p5  p6  p1 p2 p5.1 p7  p3 p7.1 p8 p4.1 p8.1

p4      0   1   1   0   0   0   1   0   0   1   0   0
p5      1   0   1   0   0   0   0   0   0   0   0   0
p6      1   1   0   1   1   1   0   1   0   0   1   0
p1      0   0   1   0   1   1   0   0   1   0   0   1
p2      0   0   1   1   0   1   0   0   1   0   0   1
p5.1    0   0   1   1   1   0   0   1   0   0   1   0
p7      1   0   0   0   0   0   0   0   0   0   0   0
p3      0   0   1   0   0   1   0   0   1   0   0   1
p7.1    0   0   0   1   1   0   0   1   0   0   1   0
p8      1   0   0   0   0   0   0   0   0   0   0   0
p4.1    0   0   1   0   0   1   0   0   1   0   0   1
p8.1    0   0   0   1   1   0   0   1   0   0   1   0
```

A dimostrazione della maggiore efficienza computazionale del secondo metodo utilizzato viene fornita la tabella 4.1. I dati si riferiscono all'esempio di dispiegamento in Figura 3.5. Questa rete può essere ottenuta correttamente con l'utilizzo di entrambi i metodi ma le differenze dal punto di vista dei tempi di calcolo sono notevoli.

f	tempi `mat_conc` $[s]$	tempi `is_concurrent` $[s]$
5	0.047	0.125
6	0.062	0.125
7	0.0911	0.312
8	0.0911	1.062
9	0.109	1.141
10	0.125	2.656
11	0.25	18.188
12	0.312	18.218
13	0.438	53.063

Tabella 4.1: Confronto dei tempi computazionali della funzione unfolding_n con l'utilizzo delle due diverse tecniche mat_conc e is_concurrent.

Il parametro `f` indica, come già spiegato al capitolo precedente, il livello di profondità del

dispiegamento. Si nota chiaramente come all'aumentare di tale parametro l'algoritmo che si basa sulla funzione `is_concurrent` fornisca i risultati in tempi decisamente meno soddisfacenti rispetto al programma che utilizza `mat_conc`. ◇

4.2 Funzione order1

Tale funzione costruisce il primo ordine di dispiegamento di una rete posto/transizione. Per riuscire ad ottenere tale prefisso è stata utilizzata la funzione `unfolding_n`, così come poc'anzi presentata, e ad essa è stata aggiunta una funzione di controllo che consente di capire se la transizione corrente è di cut-off (e quindi la procedura deve essere fermata) o meno (e dunque si può andare avanti nella costruzione del dispiegamento). A tale scopo sono state create due funzioni: una si occupa di calcolare la sequenza di scatto, mentre l'altra, a partire da tale risultato, controlla se la marcatura raggiunta è già stata visitata, consentendo così di stabilire se una transizione è di cut-off oppure no.

4.2.1 Sintassi

La funzione che costruisce il primo ordine di dispiegamento ha la seguente interfaccia:

```
[Pre_u,Post_u,p_tier,t_tier,Pexp,Texp,cut_off1,mat_conc,U]=
=order1(Pre,Post,M0)
```

Gli ingressi e le uscite hanno lo stesso significato già discusso in precedenza. In questa funzione esistono però due nuove uscite:

- `cut_off1`, che contiene le transizioni di cut-off del primo ordine di dispiegamento. Esse seguono una numerazione progressiva, non tenendo conto cioè della corrispondenza con la rete originale.
- `U`, che contiene tutte le uscite già elencate con il vantaggio di compattarle tutte in un unico array di celle e di renderle quindi disponibili sotto un'unica variabile nel momento in cui si utilizza il dispiegamento per altre analisi.

4.2.2 Descrizione del programma

Il comportamento nell'algoritmo generale di dispiegamento a questo punto si divide in due parti:

- Se la transizione $\tilde{t}$ è di cut-off si crea un nuovo vettore `Pexp1` il quale verrà utilizzato per definire `Ptot`, cioè quell'insieme a partire dal quale si determinano i posti che possono abilitare una transizione. Il vettore `Pexp1` contiene tutti i posti della rete finora trovati ad eccezione dell'insieme $\tilde{t}^\bullet$ questo perché questi nuovi posti, per come sono state definite le transizioni di cut-off, non devono abilitare nessuna ulteriore transizione.
- Se invece la transizione non è di cut-off i nuovi posti trovati si aggiungono in maniera identica sia a `Pexp` che a `Pexp1`.

La funzione che sarà utilizzata per determinare se un transizione è di cut-off o meno è `is_cutoff1`. La sua interfaccia è:

```
[r]=is_cutoff1(Pre,Post,M0,Pre_u,Post_u,t_tier,Texp)
```

I parametri in ingresso sono gli stessi già descritti come ingressi e uscite della funzione `unfolding_n`. L'uscita `r` vale:

- 1 se la transizione è di cut-off.
- 0 altrimenti.

Si noti che non viene fornito nessun parametro in ingresso che indichi esplicitamente quale transizione si stia valutando. Ciò si spiega col fatto che tutti i parametri in ingresso non sono definiti a priori ma si modificano ogni volta che si trova una nuova transizione o comunque che si completa un nuovo livello. Dunque è sempre noto qual è la transizione da analizzare in quanto, essendo l'ultima ricavata, è rappresentata dall'ultima colonna di `Pre_u` e `Post_u` o equivalentemente dall'ultimo elemento di `Texp` (o meglio dal suo indice). I punti che questa nuova funzione segue sono abbastanza semplici:

1. Si richiama la funzione `firing_seq` (verrà descritta più in dettaglio nel seguito) che fornisce l'uscita `lc` all'interno della quale sono memorizzate tutte le transizioni che devono scattare dall'inizio del dispiegamento perché la transizione corrente possa verificarsi.
2. Ad ogni transizione di `lc` (tranne quella appena analizzata) si applica nuovamente la funzione `firing_seq` in modo tale che ad ogni transizione contenuta nella variabile `lc` iniziale sia associato il suo percorso o meglio la sua sequenza di scatto.
3. Non resta che calcolare la marcatura raggiunta con lo scatto di ogni sequenza e confrontare tutte queste marcature (che sono state memorizzate in una matrice `M`) per vedere se ci sono due colonne uguali. Se ciò è vero la transizione è di cut-off e la funzione restituisce `r=1` altrimenti `r=0`.

Resta ora da descrivere come è stata ottenuta la funzione `firing_seq`. Per iniziare verrà descritta l'interfaccia

```
[lc]=firing_seq(Pre_u,Post_u,t_tier)
```

Gli ingressi hanno sempre lo stesso significato già visto nelle precedenti funzioni. Anche in questa, come nella funzione precedente, non c'è nessun parametro specifico che indichi qual è la transizione corrente, la quale viene invece ricavata dalle dimensioni di `Pre_u` e `Post_u`. L'utilizzo di `t_tier` come parametro in ingresso si rende necessario per avere una misura del livello di profondità a cui si è arrivati nel dispiegamento e per sapere quindi di quanti livelli si deve risalire nella ricerca della sequenza di scatto. Questo compito è svolto dalla funzione `previous`. Essa restituisce tutti i posti (etichettati secondo la numerazione progressiva) che, nel ramo del dispiegamento corrispondente alla sequenza di scatto corrente, vengono prima della transizione che si sta analizzando. Il passo da qui ad ottenere la sequenza di scatto è breve. E' sufficiente infatti effettuare un controllo sulla matrice `Post_u` per vedere quali sono le transizioni che precedono i posti trovati tramite `previous`.

Esempio 4.4. Si prenda in considerazione la rete posto/transizione in Figura 4.1.

L'interfaccia della funzione che verrà utilizzata è:

```
[Pre_u,Post_u,p_tier,t_tier,Pexp,Texp,cut_off1,mat_conc,U]=
=order1(Pre,Post,M0)
```

I dati in ingresso sono gli stessi utilizzati nell'Esempio 4.1, i dati in uscita sono invece:

```
Pre_u =

     1     0
     0     1
     0     1
     0     0

Post_u =

     0     0
     1     0
     1     0
     0     1

p_tier =

    [1]    [1x2 double]    [1]
```

```
t_tier =

    [1]    [2]

Pexp =

     1     2     3     1

Texp =

     1     2

cut_off1 =

     2

mat_conc =

     0     0     0     0
     0     0     1     0
     0     1     0     0
     0     0     0     0

U =

  Columns 1 through 4

    [4x2 double]    [4x2 double]    {1x3 cell}    {1x2 cell}

  Column 4   through 8

   [1x4 double]    [1x2 double]    [2]     [4x4 double]
```

Il primo ordine di dispiegamento della rete, per la procedura appena descritta, è rappresentata in Figura 4.3. ◇

4.3 Funzione order2

Tale funzione costruisce il secondo ordine di dispiegamento di una rete posto/transizione.

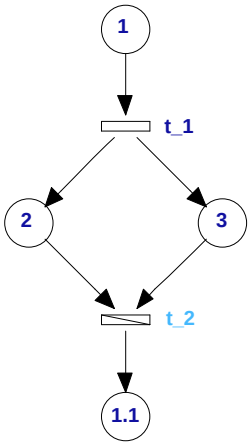

Figura 4.3: Primo ordine di dispiegamento per la rete in Figura 4.1

4.3.1 Sintassi

L'interfaccia della funzione è la seguente:

```
[Pre_u,Post_u,p_tier,t_tier,Pexp,Texp,cut_off1,cut_off2]=
=order2(Pre,Post,M0)
```

I dati in ingresso hanno lo stesso significato precedentemente discusso, mentre nei dati in uscita compare una nuova variabile, `cut_off2`, che rappresenta il vettore contenente le transizioni di cut-off del secondo ordine di dispiegamento.

4.3.2 Descrizione del programma

Le modalità di controllo che consentono di interrompere la procedura di dispiegamento nel momento in cui vengono raggiunte le transizioni di cut-off di secondo ordine sono assai simili a quelle descritte per il primo ordine di dispiegamento. Anche in questo caso si definisce un vettore `Pexp1` nel quale non verranno inseriti i posti generati dalle transizioni di cut-off del secondo ordine in modo tale che nessun'altra transizione possa essere abilitata dopo di esse.

Per non creare confusione nel seguito si indicheranno con cut-off1 le transizioni di cut-off del primo ordine di dispiegamento e con cut-off2 quelle del secondo ordine.

La funzione che sarà utilizzata per la determinazione delle transizioni di cut-off2 è `is_cutoff2` che, anche in questo caso, richiama al suo interno la funzione `firing_seq` analizzata in precedenza.

L'interfaccia di `is_cutoff2` è:

```
[r]=is_cutoff2(Pre,Post,M0,Pre_u,Post_u,Texp,t_tier,cut_off1)
```

In ingresso, oltre ai soliti parametri di cui è già stato spiegato il significato, si trova anche il vettore `cut_off1` che contiene al suo interno le transizioni di cut-off del primo ordine di dispiegamento. La loro conoscenza si rende necessaria in quanto una transizione può essere considerata *transizione specchio* solo se non appartiene al primo ordine di dispiegamento o al massimo è una transizione di `cut_off1`.

Il programma calcola come prima cosa la sequenza di scatto (`lc`) della transizione corrente e memorizza in un vettore `M` la marcatura conseguentemente raggiunta. Per ogni transizione $\tilde{t}$ contenuta in `lc` si esegue un controllo per capire se essa appartiene al primo ordine di dispiegamento. In caso contrario (o anche se è una transizione di `cut_off1`) si calcola con l'utilizzo della `firing_seq` la marcatura `MM` a cui conduce.

Se le marcature `M` e `MM` sono uguali , la transizione è di cut-off2 e viene restituito `r=1`, altrimenti la transizione non è di cut-off2 e si pone `r=0`.

Esempio 4.5. Si prenda in considerazione la rete posto/transizione in Figura 4.1.

L'interfaccia della funzione che verrà utilizzata è:

```
[Pre_u,Post_u,p_tier,t_tier,Pexp,Texp,cut_off1,cut_off2]=
 =order2(Pre,Post,M0)
```

I dati in ingresso sono gli stessi utilizzati nell'Esempio 4.1, mentre i dati in uscita sono:

```
Pre_u =

     1     0     0     0
     0     1     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
     0     0     0     1
     0     0     0     0

Post_u =

     0     0     0     0
     1     0     0     0
     1     0     0     0
     0     1     0     0
```

```
     0     0     1     0
     0     0     1     0
     0     0     0     1

p_tier =

    [1]    [1x2 double]    [1]    [1x2 double]    [1]

t_tier =

    [1]    [2]    [1]    [2]

Pexp =

     1     2     3     1     2     3     1

Texp =

     1     2     1     2

cut_off1 =

     2

cut_off2 =

     4
```

La rete di dispiegamento che si ottiene è rappresentata in Figura 4.4. ◇

4.4 Funzione McMillan

Tale funzione costruisce il prefisso finito di McMillan di una rete posto/transizione.

4.4.1 Sintassi

L'interfaccia della funzione è:

```
[Pre_u,Post_u,p_tier,t_tier,Pexp,Texp,cut_off,cut_off_ord]=
=McMillan(Pre,Post,M0)
```

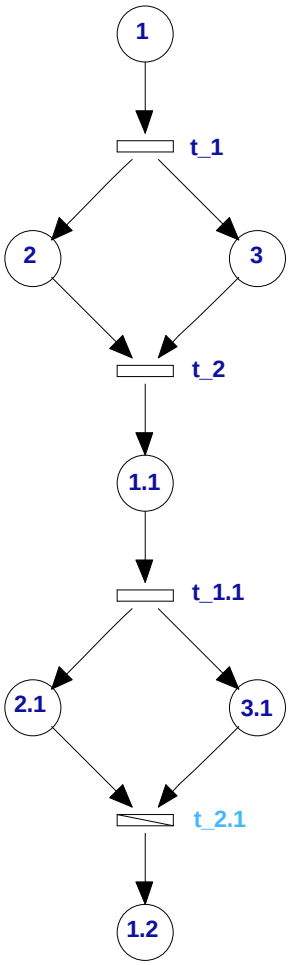

Figura 4.4: Secondo ordine di dispiegamento per la rete in Figura 4.1.

I dati in ingresso e in uscita hanno lo stesso significato discusso in precedenza, ad eccezione di `cut_off` con cui si indica il vettore contenente le transizioni di cut off secondo McMillan (`cut_off_ord` ha lo stesso significato con l'unica differenza che le transizioni sono etichettate in ordine progressivo).

4.4.2 Descrizione del programma

Si è gia visto nel capitolo precedente che la costruzione del prefisso finito di McMillan è molto simile a quella del primo ordine di dispiegamento. L'unica differenza sta nella condizione card($[\tilde{t}']$) < card($[\tilde{t}]$), ossia, la relazione tra la configurazione locale della transizione di cut-off e la configurazione locale della *transizione specchio* deve essere solo una relazione di minoranza (e non di inclusione come per il primo ordine). Il problema è stato risolto apportando una leggera modifica alla procedura del primo ordine di dispiegamento nelle modalità di riconoscimento delle transizioni di cut off. Se in quest'ultima la transizione corrente veniva controllata volta per volta per sapere se questa fosse di cut off o meno (in questo modo poteva essere soddisfatta la condizione di inclusione, ossia, la transizione specchio, se esiste, deve trovarsi per forza di cose nello stasso ramo di dispiegamento), nel prefisso finito di McMillan le transizioni invece non sono analizzate singolarmente ma viene studiato, una volta completato, l'intero livello di transizioni; è in tal modo infatti che si riesce ad avere una visione d'insieme (e non più relativa a una solo ramo) delle marcature finora raggiunte.

L'interfaccia della funzione ausiliaria che verrà utilizzata è:

```
[ctf_aus,ctf_ord_aus,r1,aus,mm11]=is_cutoffMcm(Pre_u,
Post_u,Texp,Pre,Post,M0,t_tier,m11)
```

essa richiama al suo interno le funzioni

```
[LC]=lc_McMillan(Pre_u,Post_u,t_tier)
```

```
[p]=previous(Pre_u,Post_u,posti(1),t_tier)
```

Attraverso la funzione `lc_McMillan` si risale alla precisa sequenza di scatto di tutte transizioni contenute nell'ultimo livello. Per fare ciò è stata utilizzata la funzione `previous`, che come già detto a proposito del primo ordine di dispiegamento, consente di conoscere tutti i posti appartententi al ramo che contiene la sequenza di scatto presa in considerazione. Le sequenze di scatto cosi ricavate vengono memorizzate nell'array di celle `LC`. Ad ogni cella è associata una singola sequenza di scatto e ciascuna di esse conduce a una marcatura che viene memorizzata nella matrice `M1`. Questa viene confrontata colonna per colonna con la matrice `M11` che inizialmente contiene solo marcatura `M0` e che in seguito viene aggiornata con l'aggiunta delle marcature già controllate ai livelli precedenti. Se una colonna della

matrice `M1` risulta uguale alla colonna della matrice `M11` significa che a tale colonna corrisponde una transizione di `cut_off`. Per fermare la rete di dispiegamento si adoperano le stesse modalità utilizzate per il primo ordine.

Esempio 4.6. Si è già parlato, in un esempio fatto in precedenza, delle differenze che ci possono essere tra primo ordine di dispiegamento e prefisso finito di McMillan, e in particolare di come quest'ultimo possa essere più ristretto dell'altro. Tuttavia in molti casi il prefisso finito di McMillan e il primo ordine di dispiegamento possono coincidere così come mostra la Figura 4.5 che rappresenta il prefisso finito di McMillan della rete in Figura 4.1. ◇

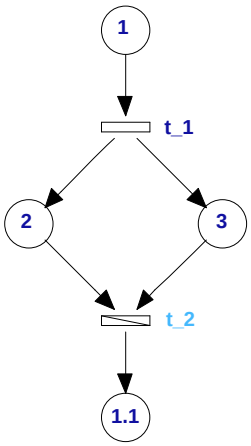

Figura 4.5: Prefisso finito di McMillan per la rete in Figura 4.1.

4.5 Funzioni unfolding_graph,graph1, graph2, graphMcM

Tali funzioni costituiscono un ausilio grafico per disegnare la rete di dispiegamento dei vari prefissi finiti.

4.5.1 Sintassi

L'interfaccia della funzione che permette di disegnare il generico dispiegamento è:

```
unfolding_graph(Pre,Post,M0,f)
```

L'interfaccia della funzione che permette di disegnare il primo ordine di dispiegamento è:

```
graph1(Pre,Post,M0)
```

L'interfaccia della funzione che permette di disegnare il secondo ordine di dispiegamento è:

`graph2(Pre,Post,M0)`

L'interfaccia della funzione che permette di disegnare il prefisso finito di McMillan è:

`graphMcM(Pre,Post,M0)`

Tutti e quattro ricevono in ingresso le matrice Pre, Post e la marcatura iniziale della rete originaria. Al loro interno contengono le funzioni descritte in precedenza che costruiscono il generico dispiegamento, il primo e il secondo ordine di dispiegamento e il prefisso finito di McMillan tramite le quali è possibile visualizzare lo schema che permetterà di disegnare le reti corrispondenti.

Esempio 4.7. Per capire meglio ciò che producono i programmi appena descritti, viene fornito un esempio di funzionamento di `graph1`. Naturalmente anche `graph2`,`graphMcM` e `unfolding_graph` utilizzano la stessa logica.

Si costruirà la rete di dispiegamento per la rete posto/transizione in Figura 4.6 (Giua e Xie, 2004).

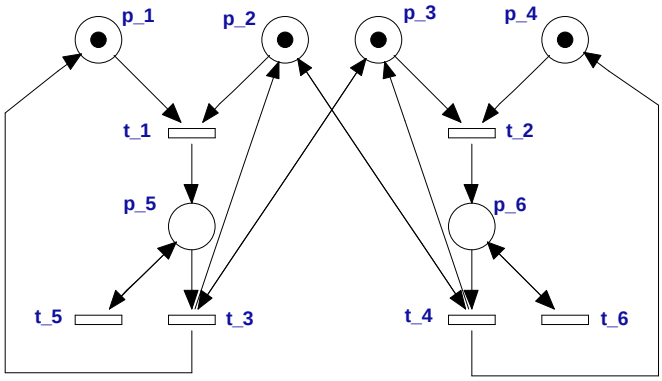

Figura 4.6: Rete posto/transizione

Avviando la funzione `graph1(Pre,Post,M0)` dal prompt dei comandi di Matlab verrà visualizzata la seguente schermata:

```
tier 0:   P1   P2   P3   P4

t1
          in:  P1   P2
         out:  P5

t2
          in:  P3   P4
         out:  P6
```

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

`tier 0`: indica quali posti ci sono nella fila corrente , in questo caso la prima fila.

Vengono poi elencate le transizioni che compongono la fila corrente e rispettivamente i posti che le abilitano (`in`) e che ne derivano (`out`). Più esplicitamente, in termini di rappresentazione grafica (Figura 4.7), ciò significa che la transizione `t1` avrà in ingresso due archi *pre*, uno dal posto `1` e l'altro dal posto `2` e in uscita un arco *post* verso il posto `5`, mentre la transizione `t2` avrà in ingresso due archi *pre*, uno dal posto `3` e l'altro dal posto `4` e in uscita un solo arco *post* verso il posto `6`:

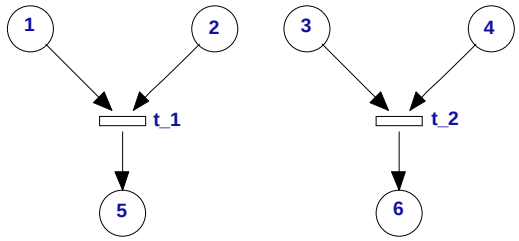

Figura 4.7: Primo passo della costruzione della rete di dispiegamento per la rete in Figura 4.6.

La procedura continua con le stesse modalità fino al termine della rete di dispiegamento:

```
tier 1:   P5   P6

t3
          in:  P3   P5
         out:  P1.1   P2.1   P3.1

t4
          in:  P2   P6
         out:  P2.2   P3.2   P4.1

t5
          in:  P5
         out:  P5.1

t6
          in:  P6
         out:  P6.1
```

```
************************************************************
tier 2:   P1.1   P2.1   P3.1   P2.2   P3.2   P4.1   P5.1   P6.1

************************************************************
cut_off: t3 t4 t5 t6
```

La rete che si ottiene è rappresentata in Figura 4.8. ◇

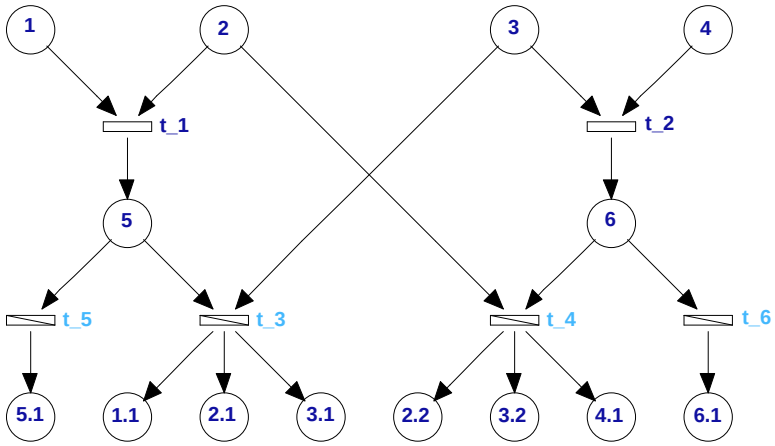

Figura 4.8: Primo ordine di dispiegamento per la rete in Figura 4.6

4.6 Funzione reach_u

Per integrare i programmi finora presentati che, come già detto, erano focalizzati a generare il dispiegamento, è stato sviluppato un programma il quale, lavorando sui risultati ottenuti, riesce ad estrapolare da essi alcune importanti informazioni sul comportamento della rete.

La funzione che verrà analizzata è `reach_u` e può essere utilizzata sostanzialmente sotto quattro differenti modalità. Il suo principale compito è quello di riconoscere se una certa marcatura in ingresso appartiene o meno allo spazio di stato della rete. Oltre a ciò, questa funzione determina anche a quali posti del dispiegamento questa marcatura corrisponde e qual è la sequenza di transizioni che devono scattare perché possa essere raggiunta.

4.6.1 Sintassi

Nel caso più generale l'interfaccia è:

```
[ret,p_corr,X]=reach_u(M0,M,U,opt)
```

Gli ingressi sono:

- `M0`: la marcatura iniziale.
- `M`: la generica marcatura di cui si vuole analizzare la raggiungibilità.
- `U`: l'uscita complessiva della funzione `order1`.
- `opt`: può assumere un valore tra 1 e 3 e serve per scegliere sotto quale modalità si vuole utilizzare il programma. In particolare:

  (a) Se opt non compare tra gli ingressi la funzione riconosce solo se la marcatura `M` è raggiungibile.

  (b) Se opt=1 la funzione fornisce in uscita anche tutti i posti del dispiegamento corrispondenti alla marcatura `M`.

  (c) Se opt=2 la funzione fornisce anche una terza uscita che indica qual è la sequenza di scatto più breve che conduce alla marcatura `M` .

  (d) Se opt=3 la terza uscita della funzione fornisce tutte le sequenze di scatto che generano `M`.

Le uscite sono invece:

- `ret`: vale 1 se la marcatura `M` è raggiungibile, 0 altrimenti.
- `p_corr`: è una matrice nelle cui righe ci sono i posti del dispiegamento che corrispondono alla marcatura `M` (`p_corr` contiene i posti corrispondenti a M nella numerazione progressiva. Per facilitare l'interpretazione dei risultati, il programma stampa su schermo anche i rispettivi posti con la numerazione della rete di partenza).
- `X`: è la terza uscita che cambia significato a seconda del valore di opt:

  (a) Se opt=2, `X` contiene la più piccola sequenza di scatto che conduce alla marcatura `M`.

  (b) Se opt=3, `X` fornisce tutte le sequenze di scatto che generano `M`.

4.6.2 Descrizione del programma

Si descriverà ora in dettaglio il comportamento del programma in ognuna delle modalità previste.

1. La prima ha come solo fine quello di stabilire se una certa marcatura è raggiungibile o meno. Il comando che si deve utilizzare per eseguire il programma sotto questa modalità è:

   `[ret]=reach_u(M0,M,U)`

   Il programma analizza innanzitutto la marcatura `M` e suddivide i posti che essa rappresenta in due insiemi:

   - `pm`: posti marcati (cioè posti in corrispondenza dei quali si trova un 1).
   - `pnm`: posti non marcati (cioè posti in corrispondenza dei quali si trova uno 0).

   Per ognuno di questi insiemi si ricavano i rispettivi posti del dispiegamento. Si genera quindi una matrice (`da_controllare`) che contiene nelle sue righe tutte le possibili combinazioni di posti del dispiegamento che corrispondono alla marcatura.

   A questo punto si deve verificare la concorrenza dei posti contenuti in ciascuna riga di `da_controllare`; per farlo si utilizza un metodo del tutto analogo a quello descritto nello sviluppo del dispiegamento che si avvale della matrice di concorrenza. Come già discusso nei precedenti paragrafi la matrice di concorrenza è una matrice simmetrica; ad ogni riga e ad ogni colonna è associato un posto del dispiegamento, gli elementi della matrice possono essere solo 1 oppure 0: si trova un 1 in corrispondenza dei posti che sono tra loro concorrenti e uno 0 in corrispondenza dei posti che non lo sono. E' chiaro come nella diagonale principale di una matrice così definita non possano esserci che zeri in quanto un posto non può per definizione mai essere concorrente con se stesso.

   Si indichi con `ex` una generica riga di `da_controllare`. I posti contenuti in tale riga devono essere tutti concorrenti tra loro dunque la sottomatrice che si estrae da mat_conc considerando solo le righe e le colonne dei posti corrispondenti a `ex` deve essere una matrice che abbia 1 dappertutto eccetto che nella diagonale principale. Se tale condizione è verificata allora i posti sono concorrenti. Questa condizione tuttavia ancora non è sufficiente. Perché la marcatura sia effettivamente raggiungibile è anche necessario che nessun altro posto sia concorrente all'insieme dei posti contenuti in `ex`. Si vuole infatti che nessun altro posto sia marcato (e se fosse concorrente sarebbe anche marcato) eccetto quelli contenuti in `ex` che corrispondono ai `pm`. Si esegue perciò un controllo per verificare che nessuno dei posti del dispiegamento corrispondenti all'insieme `pnm` sia concorrente ai posti di `ex`. Se anche questa condizione è verificata allora l'uscita diventa `ret=1` e si esce dal programma.

2. La seconda modalità di utilizzo è praticamente identica alla prima; l'unica differenza consiste nel fatto che il programma non termina la ricerca appena riconosce che una marcatura è raggiungibile ma continua ad analizzare tutti gli insiemi di posti del dispiegamento (le righe di `da_controllare` di cui si è parlato al punto1) e oltre a indicare attraverso l'uscita `ret` se la marcatura `M` è raggiungibile, fornisce anche

l'uscita `p_corr`. Quest'ultima è una matrice nelle cui righe si trovano i posti del dispiegamento corrispondenti a `M`. Per facilitare la lettura, tali posti vengono anche stampati su schermo con la numerazione della rete di partenza. Se si vuole utilizzare il programma in questa modalità si deve utilizzare l'interfaccia:

```
[ret,p_corr]=reach_u(M0,M,U,1)
```

3. Una terza possibiltà offerta da questa funzione è quella di conoscere la più breve sequenza di scatto che porta alla marcatura `M` desiderata. Per poter accedere a quest'informazione si deve chiamare la funzione con la seguente sintassi:

   ```
   [ret,p_corr,X]=reach_u(M0,M,U,2)
   ```

   Questa parte del programma calcola la sequenza di scatto a partire dalla matrice `p_corr` descritta al punto precedente. In particolare si analizza ogni riga di tale matrice e per ognuna di esse si procede alla seguente analisi:

   - Per ogni posto contenuto nella riga corrente si determina qual è la transizione `tra` che lo genera.
   - Si calcola la sequenza di scatto relativa a `tra` e la si memorizza in `lc`.
   - A questo punto le transizioni di `lc` vengono separate a seconda del livello a cui appartengono; la struttura che si ottiene si chiama `lc_tier` e contiene nelle sue celle le transizioni della sequenza di scatto corrispondenti ad ogni singolo livello. Quando tutti i posti della riga di `p_corr` saranno stati controllati `lc_tier` conterrà tutte le transizioni che conducono alla marcatura desiderata divise a seconda del livello a cui appartengono.
   - Processata ogni riga di `p_corr` è sufficiente un confronto tra tutti i risultati raggiunti per capire qual è la sequenza di scatto più breve.

4. L'ultima modalità disponibile infine consente di ottenere non solo la più breve, ma tutte le sequenze di scatto associate a tutti gli insiemi di posti che nel dispiegamento rappresentano la marcatura. L'interfaccia da utilizzare in questo caso è :

   ```
   [ret,p_corr,X]=reach_u(M0,M,U,3)
   ```

   Sia in questo caso che in quello precedente la funzione è programmata per stampare su schermo le varie sequenze di scatto con la numerazione della rete iniziale al fine di facilitare l'interpretazione dei risultati ottenuti.

Un esempio chiarirà meglio il funzionamento della procedura descritta.

Esempio 4.8. Si consideri la rete di Petri in Figura 4.9 (He e Lemmon, 2002),

e il suo dispiegamento in Figura 4.10.

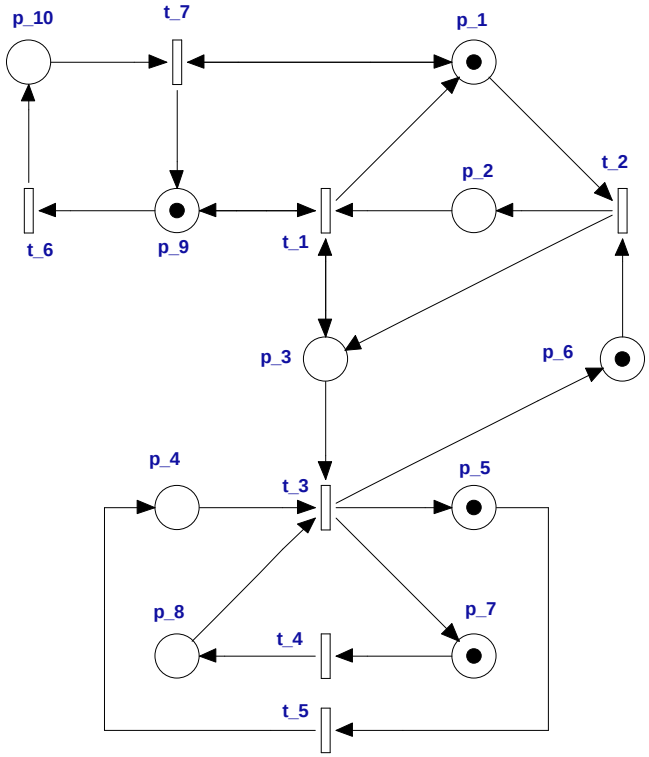

Figura 4.9: Rete posto/transizione

Si voglia utilizzare la funzione sulla marcatura iniziale $M0 = [1000111010]$. Matlab fornisce per ognuna delle 4 modalità i seguenti risultati:

1. `[ret]=reach_u(M0,M,U)`

```
ret =

    1
```

2. `[ret,p_corr]=reach_u(M0,M,U,1)`

```
1^ set of places:  p1   p5   p6   p7   p9
2^ set of places:  p1.1 p5.2 p6.2 p7.2 p9.1
3^ set of places:  p1.2 p5   p6   p7   p9.2
4^ set of places:  p1.3 p5.2 p6.2 p7.2 p9.3

ret =

    1
```

```
p_corr =

     1     2     3     4     5
    11    19    20    21    13
    17     2     3     4    18
    25    19    20    21    26
```

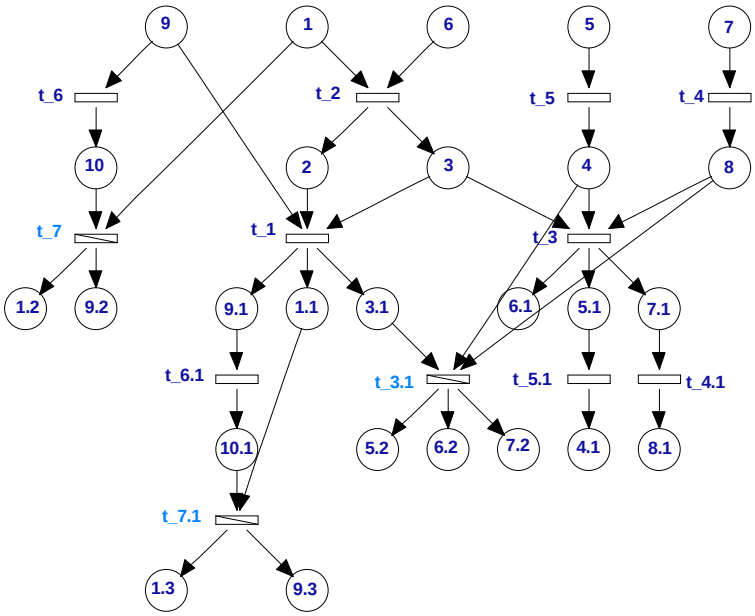

Figura 4.10: Dispiegamento della rete posto/transizione in 4.9

In effetti se si guarda la rete tutte le configurazioni di posti segnate sono concorrenti e corrispondono alla marcatura iniziale. In `p_corr` sono memorizzati tutti i posti considerando la numerazione progressiva del dispiegamento.

3. `[ret,p_corr,X]=reach_u(M0,M,U,2)`

```
the shortest firing sequence is:

ret =

     1
```

```
p_corr =

     1     2     3     4     5
    11    19    20    21    13
    17     2     3     4    18
    25    19    20    21    26

X =

     []
```

Visto che la marcatura in questione coincide con la marcatura iniziale è corretto che la sequenza di scatto più breve sia quella vuota.

4. `[ret,p_corr,X]=reach_u(M0,M,U,3)`

```
 1^ set of places:  p1    p5    p6    p7    p9
 2^ set of places:  p1.1  p5.2  p6.2  p7.2  p9.1
 3^ set of places:  p1.2  p5    p6    p7    p9.2
 4^ set of places:  p1.3  p5.2  p6.2  p7.2  p9.3


 the 1 firing sequence is:
 the 2 firing sequence is:  t2  t4  t5  t1  t3.1
 the 3 firing sequence is:  t6  t7
 the 4 firing sequence is:  t2  t4  t5  t1  t3.1 t6.1 t7.1

ret =

     1

p_corr =

     1     2     3     4     5
    11    19    20    21    13
    17     2     3     4    18
    25    19    20    21    26

X =

     []    [1x5 double]    [1x2 double]    [1x7 double]
```

Le quattro sequenze di scatto sono relative ai quattro insiemi di posti di `p_corr`. ◇

Capitolo 5

Analisi dei risultati raggiunti

Questo capitolo sarà dedicato alla presentazione dei risultati raggiunti. In particolare verrà sottolineata la maggiore efficienza computazionale dell'analisi basata sul dispiegamento rispetto alla tradizionale analisi che si avvale del grafo di raggiungibilità. Saranno quindi prospettati una serie di esempi volti a dimostrare le notevoli differenze che si riscontrano nei tempi attesi per ottenere i risultati attraverso i due diversi approcci. Le due distinte procedure sono state applicate a reti posto/transizione costituite da più cellule identiche collegate tra loro. Questo consente di valutare le prestazioni su sistemi nei quali aumenta la concorrenzialità e che pertanto si prestano particolarmente bene a un'analisi mediante dispiegamento.

5.1 I filosofi cinesi

Verrà trattato il problema dei *filosofi cinesi* già presentato al Capitolo 2. La rete di base, come già detto, potrà essere ampliata e potrà variare in funzione di due parametri n e m.

- n: rappresenta il numero di filosofi a tavola.
- m: rappresenta il numero di stati in cui può trovarsi il singolo filosofo.

Ovviamente anche il numero delle bacchette sarà sempre pari a n. Un possibile esempio di tale sistema è raffigurato in Figura 5.1 per n=8. .

In Figura 5.2 è rappresentata la rete posto/transizione che descrive il sistema per $n = 3$ e $m = 4$ mentre in Figura 5.3 è rappresentato il suo dispiegamento. Per esigenze grafiche è stato omesso il grafo di raggiungibilità che contiene 36 marcature.

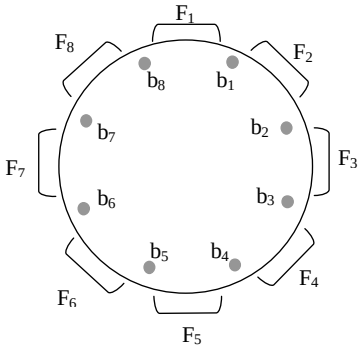

Figura 5.1: Sistema con 8 filosofi

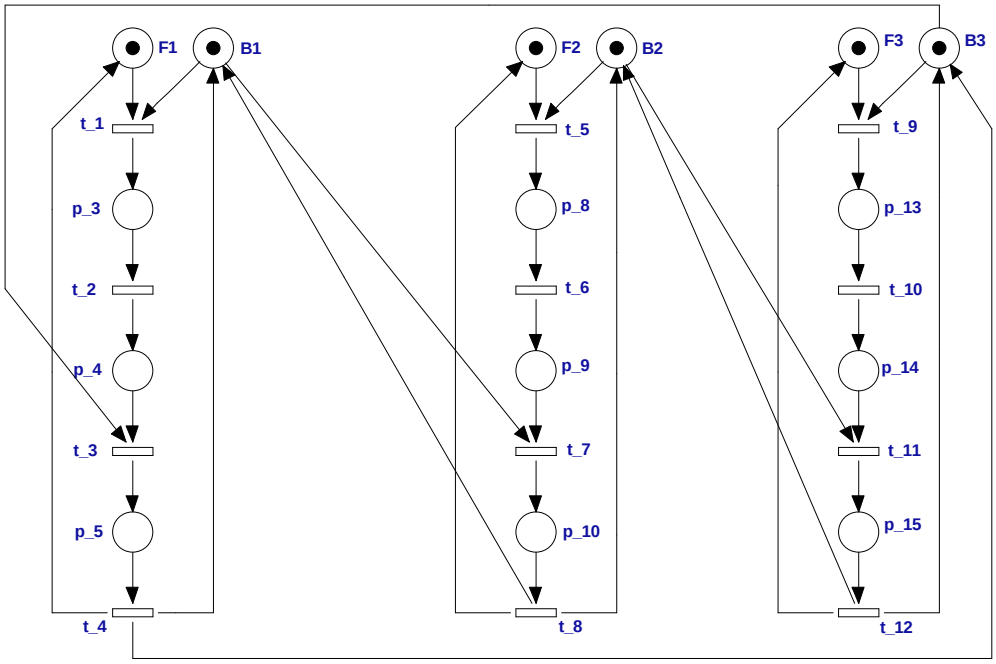

Figura 5.2: Rete posto/transizione equivalente al sistema con 3 filosofi e 4 stati

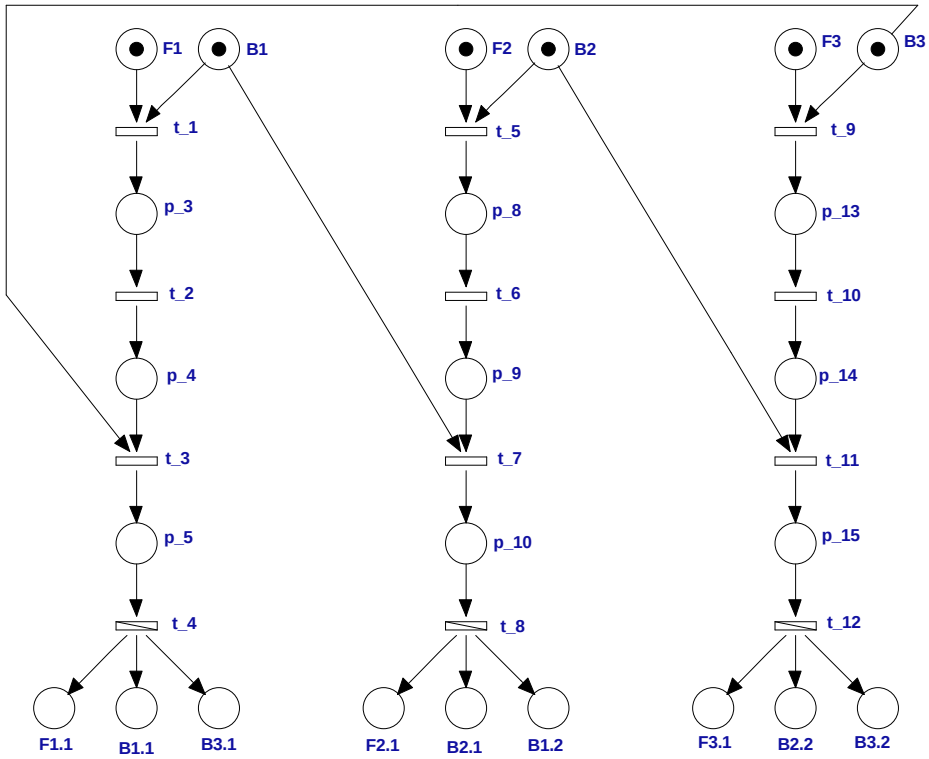

Figura 5.3: Dispiegamento della rete in Figura 5.2

Nella Tabella 5.1 sono riportati i tempi di computazione impiegati dalla funzione `order1` al variare del numero dei filosofi e del numero degli stati in cui ognuno di essi si può trovare.

Nella Tabella 5.2 sono riportati il numero di posti e il numero di transizioni del primo ordine di dispiegamento al variare del numero dei filosofi (n) e del numero degli stati (m) in cui ognuno di essi si può trovare.

Nella Tabella 5.3 sono riportati i tempi di computazione impiegati dalla funzione `order2` al variare di n ed m.

Nella Tabella 5.4 sono riportati il numero di posti e il numero di transizioni del secondo ordine di dispiegamento al variare di n ed m.

I risultati ottenuti utilizzando il metodo del dispiegamento sono stati messi a confronto con quelli ottenuti tramite l'utilizzo del grafo di raggiungibilità. A tal fine è stata usata la funzione `graph` che fa parte di un Toolbox di reti di Petri sviluppato presso il Dipartimento di Ingegneria Elettrica ed Elettronica dell'Università degli Studi di Cagliari disponibile sul sito `http://www.diee.unica.it/~giua/ARP` Nella Tabella 5.5 sono riportati i tempi di computazione impiegati dalla funzione `graph` al variare di n ed m.

Nella Tabella 5.6 è riportato il numero di marcature raggiungibili al variare di n ed m.

TEMPI [s]	3 stati	4 stati	5 stati	6 stati	7 stati	8 stati	9 stati	10 stati
2 filosofi	0.031	0.031	0.047	0.093	0.078	0.125	0.125	0.157
3 filosofi	0.047	0.047	0.078	0.125	0.141	0.203	0.187	0.25
4 filosofi	0.078	0.109	0.125	0.14	0.218	0.297	0.281	0.344
5 filosofi	0.078	0.109	0.172	0.203	0.312	0.375	0.391	0.469
6 filosofi	0.125	0.141	0.172	0.265	0.391	0.437	0.468	0.594
7 filosofi	0.156	0.234	0.25	0.344	0.453	0.593	0.703	0.765
8 filosofi	0.187	0.172	0.235	0.328	0.5	0.593	0.781	0.938
9 filosofi	0.218	0.297	0.328	0.5	0.781	0.984	0.922	1.093
10 filosofi	0.203	0.203	0.313	0.578	0.844	1.015	1.234	1.312

Tabella 5.1: Tempi computazionali della funzione order1 al variare di n ed m.

$p \times t$	3 stati	4 stati	5 stati	6 stati	7 stati	8 stati	9 stati	10 stati
2 filosofi	14×6	16×8	18×10	20×12	22×14	24×16	26×18	28×20
3 filosofi	21×9	24×12	27×15	30×18	33×21	36×24	39×27	42×30
4 filosofi	28×12	32×16	36×20	40×24	44×28	48×32	52×36	56×40
5 filosofi	35×15	40×20	45×25	50×30	55×35	60×40	65×45	70×50
6 filosofi	42×18	48×24	54×30	60×36	66×42	72×48	78×54	84×60
7 filosofi	49×21	56×28	63×35	70×42	77×49	84×56	91×63	98×70
8 filosofi	56×24	64×32	72×40	80×48	88×56	96×64	104×72	112×80
9 filosofi	63×27	72×36	81×45	90×54	99×63	108×72	117×81	126×90
10 filosofi	70×30	80×40	90×50	100×60	110×70	120×80	130×90	140×100

Tabella 5.2: Numero di Posti (p) e numero di Transizioni (t) nel primo ordine di dispiegamento $(p \times t)$ al variare di m ed n

TEMPI [s]	3 stati	4 stati	5 stati	6 stati	7 stati	8 stati	9 stati	10 stati
2 filosofi	0.125	0.219	0.328	0.5	0.64	0.875	1.14	1.468
3 filosofi	0.265	0.406	0.625	0.875	1.234	1.594	2.093	2.719
4 filosofi	0.5	0.766	1.125	1.547	2.141	3.032	3.718	4.937
5 filosofi	0.688	1.078	1.562	2.265	3.297	4.344	5.75	7.453
6 filosofi	0.906	1.453	2.25	3.375	4.672	6.687	8.656	11.25
7 filosofi	1.25	2.219	3.235	4.907	7.282	9.766	12.907	17.031
8 filosofi	1.734	2.89	4.765	7.219	10.359	14.312	19.422	25.672
9 filosofi	2.735	4.343	7	10.921	15.125	22.45	29.969	37.63
10 filosofi	3.547	6.297	9.89	15.676	23.375	32.61	42.609	54.266

Tabella 5.3: Tempi computazionali della funzione order2 al variare di n ed m.

$p \times t$	3 stati	4 stati	5 stati	6 stati	7 stati	8 stati	9 stati	10 stati
2 fil	34×18	40×24	46×30	52×36	58×42	64×48	70×50	76×60
3 fil	66×33	72×42	81×51	90×60	99×69	108×78	117×87	126×96
4 fil	100×52	112×64	124×76	136×88	148×100	160×112	172×124	184×136
5 fil	125×65	140×80	155×95	170×110	185×125	200×140	215×155	230×170
6 fil	150×78	168×96	186×114	204×132	222×150	240×168	258×186	276×204
7 fil	175×91	196×112	217×133	238×154	295×175	280×196	301×217	322×238
8 fil	200×104	224×128	248×152	272×176	296×200	320×224	344×248	368×272
9 fil	225×117	252×144	279×171	306×198	333×225	360×252	387×279	414×306
10 fil	250×130	280×160	310×190	340×220	370×250	400×280	430×310	460×340

Tabella 5.4: Numero di Posti (p) e numero di Transizioni (t) nel secondo ordine di dispiegamento $(p \times t)$ al variare di n ed m.

TEMPI $[s]$	3 stati	4 stati	5 stati	6 stati	7 stati	8 stati	9 stati	10 stati
2 filosofi	0.016	0.078	0.015	0.031	0.093	0.125	0.157	0.265
3 filosofi	0.016	0.109	0.312	0.61	1.828	4.329	10.78	21.469
4 filosofi	0.063	0.672	4.547	23.188	95.89	343.031	..	..
5 filosofi	0.328	7.453	98.641	..	..	..	..	..
6 filosofi	2.078	101.71	..	..	..	..	..	..
7 filosofi	14.391	..	..	..	..	..	..	..
8 filosofi	99.109	..	..	..	..	..	..	..
9 filosofi	724.65	..	..	..	..	..	..	..
10 filosofi	..	..	..	..	..	..	..	..

Tabella 5.5: Tempi computazionali della funzione graph al variare di n ed m. Il simbolo $(..)$ indica che i tempi computazionali necessari all'ottenimento di questi risultati sono eccessivamente elevati.

M	3 stati	4 stati	5 stati	6 stati	7 stati	8 stati	9 stati	10 stati
2 filosofi	6	11	18	27	38	51	66	83
3 filosofi	14	36	76	140	234	364	536	756
4 filosofi	34	119	322	727	1442	2599	..	..
5 filosofi	82	393	1364	..	..	..	..	..
6 filosofi	198	1298	..	..	..	..	..	..
7 filosofi	478	..	..	..	..	..	..	..
8 filosofi	1154	..	..	..	..	..	..	..
9 filosofi	2786	..	..	..	..	..	..	..
10 filosofi	..	..	..	..	..	..	..	..

Tabella 5.6: Numero di marcature raggiungibili (M) al variare di n ed m. Il simbolo $(..)$ indica che i tempi computazionali necessari all'ottenimento di quel risultato sono eccessivamente elevati.

I grafici in Figura 5.4, 5.5, 5.6, 5.7, 5.8, 5.9 rappresentano l'andamento dei tempi computazionali di alcuni significativi esempi estrapolati dalle Tabelle 5.1 e 5.5.

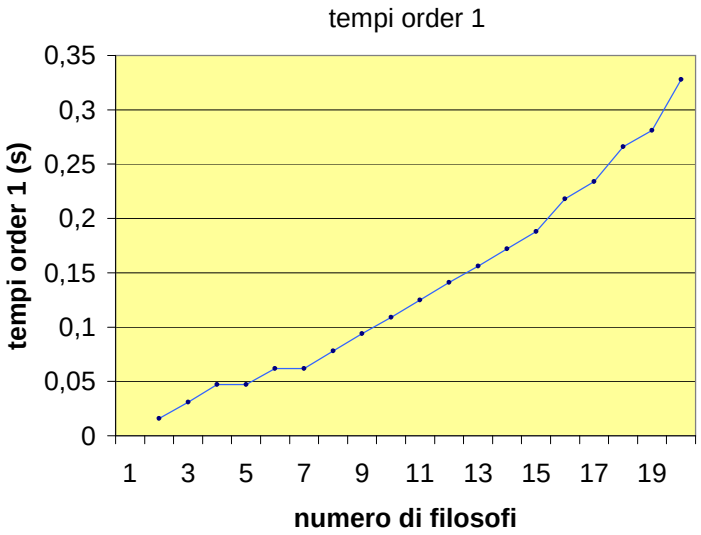

Figura 5.4: Andamento dei tempi computazionali della funzione order1 con $m = 4$ e n variabile

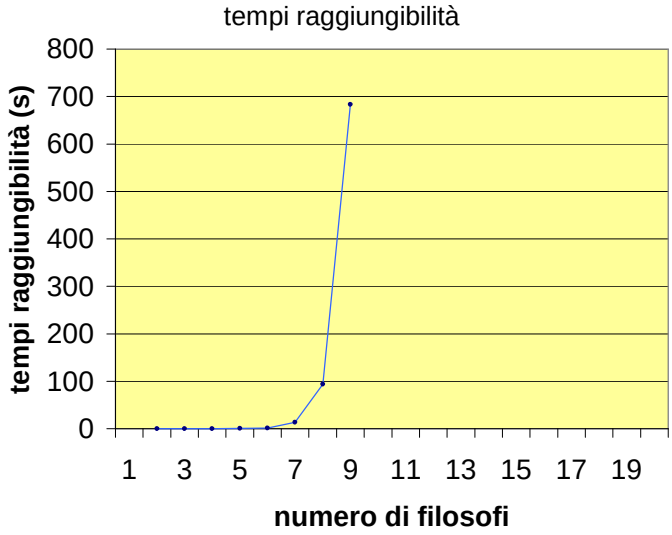

Figura 5.5: Andamento dei tempi computazionali della funzione graph per il calcolo del grafo di raggiungibilità con $m = 4$ e n variabile

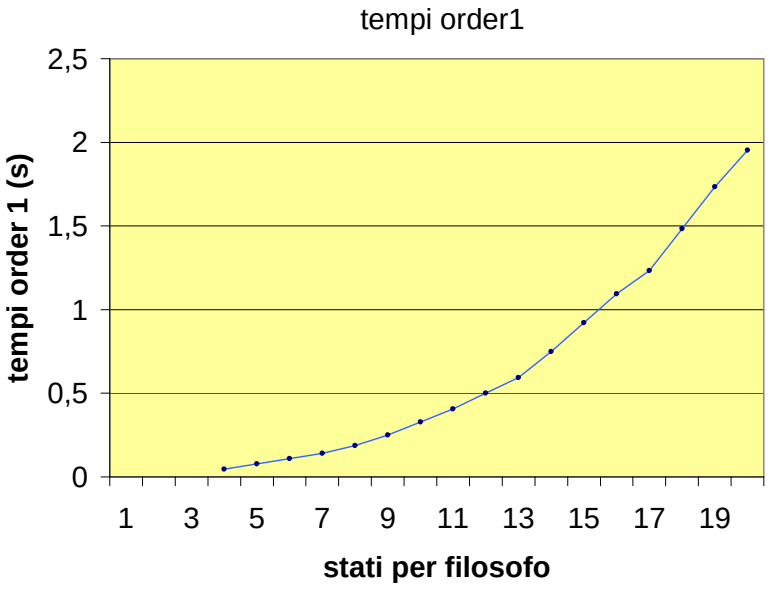

Figura 5.6: Andamento dei tempi computazionali della funzione order1 con $n = 5$ e m variabile.

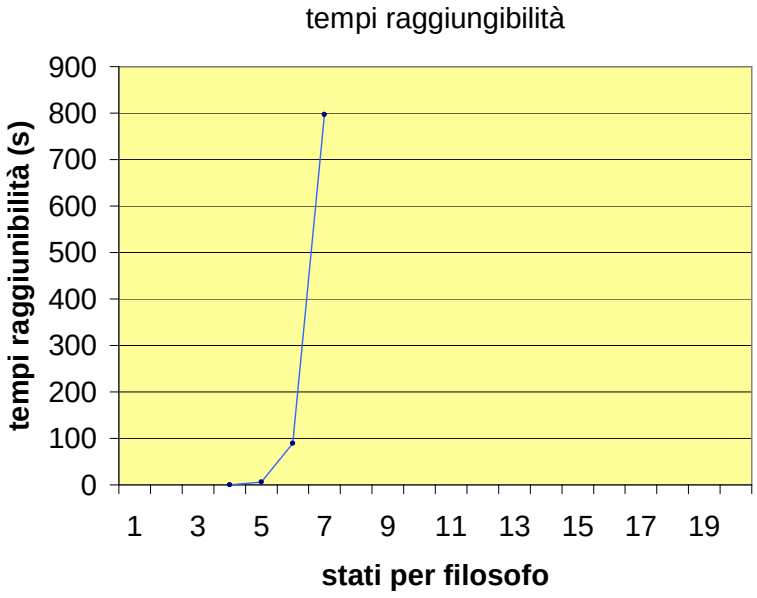

Figura 5.7: Andamento dei tempi computazionali della funzione graph per il calcolo del grafo di raggiungibilità con $n = 5$ e m variabile.

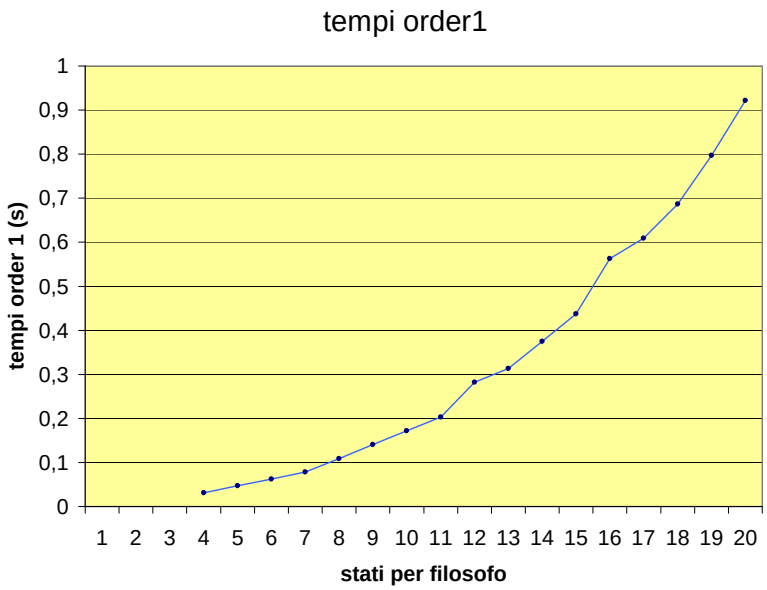

Figura 5.8: Andamento dei tempi computazionali della funzione order1 con $n = 3$ e m variabile

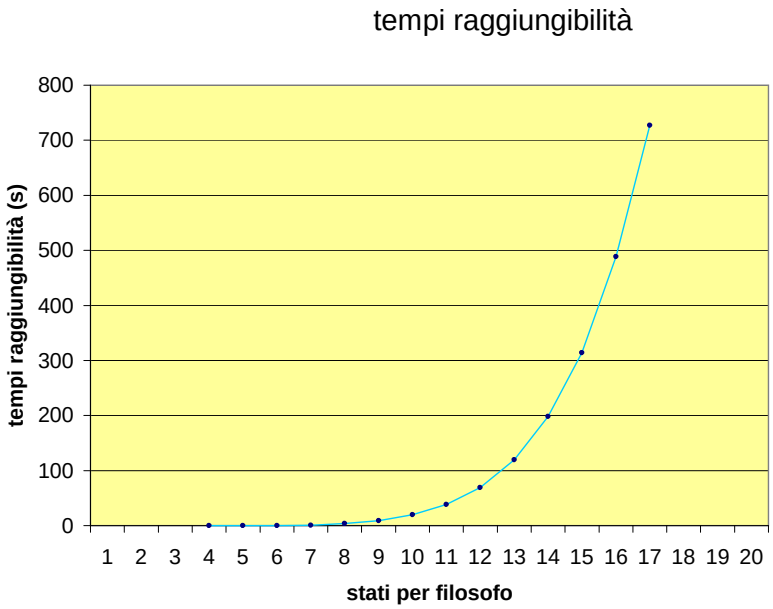

Figura 5.9: Andamento dei tempi computazionali della funzione graph per il calcolo del grafo di raggiungibilità con $n = 3$ e m variabile.

5.2 Slotted ring protocol

Con questo termine si indica una politica di gestione delle trasmissioni utilizzata nel campo delle telecomunicazioni nel caso in cui molti sistemi si trovino connessi per scambiare informazioni. Il protocollo *slotted ring* è a gestione casuale e ogni sistema cerca di trasmettere quando vuole, verificando che il supporto trasmissivo sia libero. Questo sistema, usato nelle topologie ad anello, riceve e ritrasmette un messaggio di lunghezza fissa (slot).

Nella Figura 5.10 (Esparza, Römer e Vogler, 1996) viene rappresentato la rete di posto/transizione equivalente al modello appena presentato nel caso in cui si trovino connessi due sistemi.

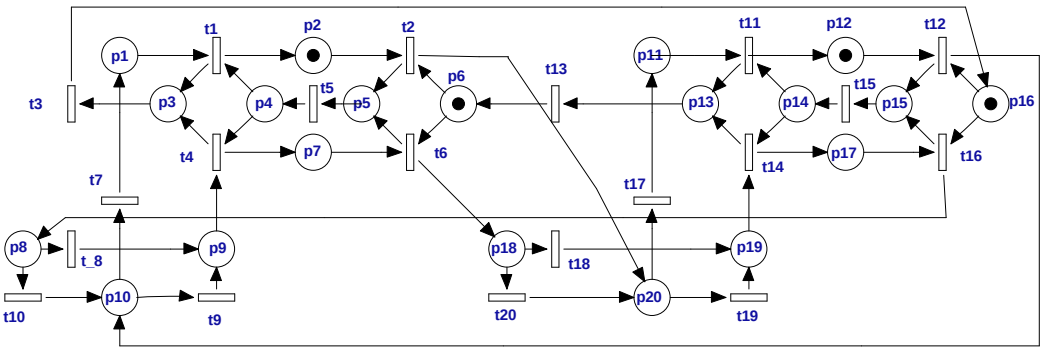

Figura 5.10: Slotted ring protocol per due sistemi.

Nella Tabella 5.7 viene presentato un confronto tra i risultati in termini di complessità computazionale raggiunti attraverso l'utilizzo delle funzioni `McMillan` e `graph` al variare del numero N dei sistemi connessi in trasmissione.

N	Tempi `McMillan` [s]	Tempi `graph` [s]	t	p	M
1	0.117	0.032	12	18	12
2	0.563	3.453	68	100	208
3	6.219	1520	288	414	4032
4	349	..	1248	1812	..

Tabella 5.7: Confronto tra le funzioni McMillan e graph: tempi computazionali, numero di transizioni (t) e di posti (p) del dispiegamento, numero di marcature raggiungibili (M). Il simbolo ($..$) indica che i tempi computazionali necessari all'ottenimento di quel risultato sono eccessivamente elevati.

Appendice A

Glossario

- **Relazione d'ordine**:
  - **Ordine Totale**: Dato un insieme W, si dice che per esso vale una relazione di ordine totale se è sempre possibile stabilire una relazione d'ordine tra una qualsiasi coppia dei suoi elementi. Un esempio di ordine totale è costituito dalla relazione $\leq$ tra i punti di una retta. Come mostra la Figura A.1, data una retta r e due punti $(A, B) \in r$ si può stabilire senza ambiguità che $A \leq B$.

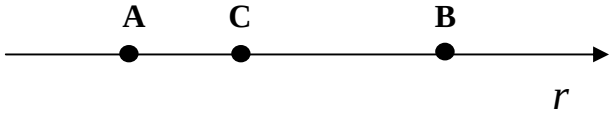

Figura A.1: Relazione di ordine totale: punti di una retta.

Se si volesse generalizzare questo concetto e applicare una relazione di ordine $\leq$ in un piano, non si potrebbe più parlare di ordine totale. Le coordinate in tal caso sono infatti due e non è possibile determinare una relazione di ordine $\leq$ che valga per una qualsiasi coppia dei punti del piano. Si rende perciò necessario introdurre una nozione di ordine meno forte che prende il nome di *ordine parziale*.

  - **Ordine Parziale**: Dato un insieme W, si dice che per esso vale una relazione di ordine parziale se non è possibile stabilire una relazione di ordine tra una qualsiasi coppia dei suoi elementi. Una relazione di ordine parziale è transitiva, antisimmetrica, riflessiva. In Figura A.2 (a) si può dire con certezza che il punto A è minore di tutti gli altri (in quanto entrambe le sue coordinate sono minori di tutte le altre), è impossibile invece stabilire la relazione d'ordine che intercorre tra il punto B e il punto C. La Figura A.2 (b) mostra invece le relazioni che possono essere definite tra i punti della Figura A.2 (a). Uno schema di tale tipo rappresenta appunto una relazione di ordine parziale.

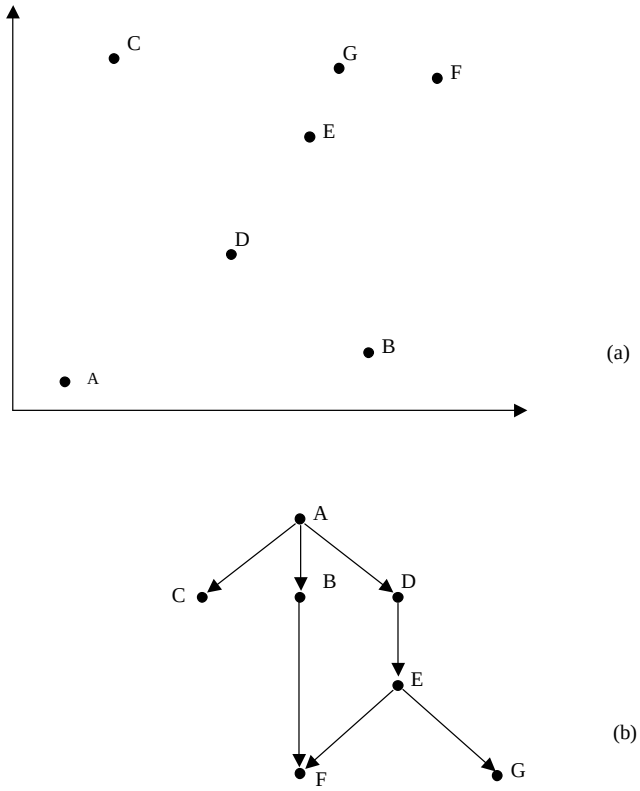

Figura A.2: Relazione di ordine parziale: punti di un piano.

- **Occorrenza**

  Nel linguaggio di molte scienze (statistica, logica matematica, linguistica, ecc.), ognuna delle volte in cui un determinato fenomeno si verifica o un dato elemento compare; ricorrenza.

  (Battaglia, UTET)

- **Dispiegamento**:

  Il dispiegare, il distendere.

  (Battaglia, UTET)

- **Dispiegare**:

  1. Distendere ciò che era piegato; stirare, spianare in modo da offrire la superficie interamente alla vista; 2 Allargare, svolgere, sviluppare, espandere; 7 Far conoscere, esporre; far comprendere, rendere chiaro e intellegibile, spiegare, chiarire; 12 Svolgersi, attuarsi, operare secondo determinate regole, assumendo certe caratteristiche (un fenomeno, un'attività); verificarsi, svilupparsi, prendere corpo.

  (Battaglia, UTET)

Appendice B

Listati dei programmi

B.1 Funzione unfolding_n

```
function[Pre_u,Post_u,p_tier,t_tier,Pexp,Texp]=
=unfolding_n(Pre,Post,M0,f)

M0=M0';
Texp=[];
t_tier(1)={1};
[m,n]=size(Pre);
P0=find(M0);
p_tier(1)={P0};
cc=length(P0);
tt=0; i=0;
kkk=0;
Pexp=P0;
Pexp1=P0;
Pi_ord=[];
P0ord=[];
P1ord=[];
Pre_u=[];
Post_u=[];
P1=[];
P11=[];
mat_conc=ones(cc);
```

```
for lo=1:cc
    mat_conc(lo,lo)=0;
end


for kk=1:length(P0)
    P0ord=[P0ord kk];
end

Pi_old_ord=P0ord;
Pexp_ord=P0ord;

for g=1:f
    pp=1;
    P11s={};
    tr=[];
    for i=1:n
        stessi={};

        Ptot=[];
        Ptot_ord=Pexp_ord;


        for j=1:length(Pexp_ord)
            for k=1:length(Pi_ord)
                if Pexp_ord(j)==Pi_ord(k)

                    Ptot_ord(j)=[0];
                end
            end
        end
        [u, v, Ptot_ord]=find(Ptot_ord);


        for ii=1:length(Ptot_ord)
            in=Ptot_ord(ii);

            Ptot=[Ptot Pexp(in)];
        end

        pre_current=Pre(:,i);
        ind=find(pre_current);
        stessi={};
```

```
for g12=1:length(ind)
    stes=[];
    for gg12=1:length(Ptot)
        if ind(g12)==Ptot(gg12)
            stes=[stes gg12];
        end
    end
    stessi(g12)={stes};


end
la=cell2mat(stessi(1))';

x=length(la);
for i12=1:length(stessi)-1
    a2=[];
    b=cell2mat(stessi(i12+1));
    for k12=1:x
        for j12=1:length(b)
            a3=[ la(k12,:) b(j12)];
            a2=[a2;a3];
        end
    end
    la=a2;
    [x xx]=size(la);
end

PPsubtot=la;

[mm,nn]=size(PPsubtot);


for z=1:mm
    h=PPsubtot(z,:);

    if ~isempty(find(h))

        A=[];

        for p=1:length(h)
            for q=1:length(Pi_old_ord)
                if (h(p)==Pi_old_ord(q))
```

```
            A=[A h(p)];
        end
    end
end

if ~isempty(A);
    a=[];

  lh=length(h);
   for j=1:lh
    for jj=1:lh
     if (j~=jj)&&mat_conc(h(j),h(jj))==1
       a=[a 1];
     end
    end
   end
    vari=lh*lh-lh;
    if length(a)==vari


      tr=[tr i];
      Texp=[Texp i];
      tt=tt+1;

      P11=find(Post(:,i));


      P1=[P1 P11'];
      if ~isempty(P11)
          P11s(pp)={P11'};
          pp=pp+1;
      end
      for kk=kk+1:length(P11)+kk
          P1ord=[P1ord kk];
      end

      for t=1:length(h)
          Pre_u(h(t),tt)=1;
      end

      for kkk=kkk+1:length(P11)+kkk
          Post_u(P1ord(kkk),tt)=1;
      end
```

```
                        Pexp=[Pexp P11'];


                        for bb=length(Pexp_ord)+1:length(Pexp)
                            Pexp_ord=[Pexp_ord bb];
                        end


                        for cc=cc+1:cc+length(P11)
                            Pi_ord=[Pi_ord cc];
                        end

                    end
                end
            end
        end

end
t_tier(g)={tr};

p_tier(g+1)={P1};

kkk=0;
g=g+1;
mat_conc1=mat_conc;
[mm1 nn1]=size(mat_conc1);

for vo1=1:length(P1ord)

    tra=find(Post_u(P1ord(vo1),:));
    pos=find(Pre_u(:,tra));
    matri=[];
    for op=1:length(pos)
        matri=[matri;mat_conc(pos(op),:)];
        if op>1
            matri=matri(1,:)&matri(2,:);
        end
    end

    mat_conc2(vo1,:)=matri;
end
mat_conc3=mat_conc2';
```

```
mat_conc4=zeros(length(P1ord));

for vvo1=1:length(P1ord)

    ttra=find(Post_u(P1ord(vvo1),:));
    ppos=find(Pre_u(:,ttra));
    mmatri=[];
    for oop=1:length(ppos)
        mmatri=[mmatri;mat_conc3(ppos(oop),:)];
        if oop>1
            mmatri=mmatri(1,:)&mmatri(2,:);
        end
    end

    mat_conc4(vvo1,:)=mmatri;
end
lung1=0;
lung=0;
for mlp=1:length(P11s)
  mom=cell2mat(P11s(mlp));

  lung1=lung1+lung;
  lung=length(mom);


  if lung>1
    matd1=ones(lung);

   for lo=1:lung
       matd1(lo,lo)=0;
   end
   mat_conc4(lung1+1:lung1+lung,lung1+1:lung1+lung)=matd1;
  end
end

mat_conc=[mat_conc1 mat_conc3;mat_conc2 mat_conc4];


Pi_ord=[];
Pi_old=P1;
Pi_old_ord=P1ord;
P1=[];
P11=[];
```

```
    P1ord=[];
    kkk=0;

    mat_conc1=[];
    mat_conc2=[];
    mat_conc3=[];
    mat_conc4=[];

end

if ~isequal(size(Pre_u),size(Post_u))
    [m,n]=size(Pre_u);
    [m1,n1]=size(Post_u);

    for kk=m+1:m1
        Pre_u(kk,:)=0;
    end
end
```

B.2 Funzione order1

```
function[Pre_u,Post_u,p_tier,t_tier,Pexp,Texp,cut_off1,U]=
=order1(Pre,Post,M0)

M0=M0';
Texp=[];
t_tier(1)={1};
[m,n]=size(Pre);
P0=find(M0);
p_tier(1)={P0};
cc=length(P0);
tt=0; i=0;
kkk=0;
Pexp=P0;
Pexp1=P0;
Pi_ord=[];
P0ord=[];
P1ord=[];
Pre_u=[];
```

```
Post_u=[];
P1=[];
P11=[];
cut_off1=[];
mat_conc=ones(cc);

for lo=1:cc
    mat_conc(lo,lo)=0;
end

for kk=1:length(P0)
    P0ord=[P0ord kk];
end

Pi_old_ord=P0ord;
Pexp_ord=P0ord;


g=1;
while ~isempty(cell2mat(p_tier(g)))

    pp=1;
    P11s={};
    tr=[];
    for i=1:n

        Ptot=[];
        Ptot_ord=Pexp_ord;


        for j=1:length(Pexp_ord)
            for k=1:length(Pi_ord)
                if Pexp_ord(j)==Pi_ord(k)

                    Ptot_ord(j)=[0];
                end
            end
        end
        [u, v, Ptot_ord]=find(Ptot_ord);


        for ii=1:length(Ptot_ord)
            in=Ptot_ord(ii);
```

```
    Ptot=[Ptot Pexp1(in)];
end
stessi={};
pre_current=Pre(:,i);
ind=find(pre_current);
for g12=1:length(ind)
    stes=[];
    for gg12=1:length(Ptot)
        if ind(g12)==Ptot(gg12)
            stes=[stes gg12];
        end
    end
    stessi(g12)={stes};
end

la=cell2mat(stessi(1))';

x=length(la);
for i12=1:length(stessi)-1
    a2=[];
    b=cell2mat(stessi(i12+1));
    for k12=1:x
        for j12=1:length(b)
            a3=[ la(k12,:) b(j12)];
            a2=[a2;a3];
        end
    end
    la=a2;
    [x xx]=size(la);
end

PPsubtot=la;


[mm,nn]=size(PPsubtot);

for z=1:mm
    h=PPsubtot(z,:);
    if ~isempty(find(h))

        A=[];
```

```
for p=1:length(h),
    for q=1:length(Pi_old_ord)
        if (h(p)==Pi_old_ord(q))
            A=[A h(p)];
        end
    end
end

if ~isempty(A);

    a=[];
    lh=length(h);
    for j=1:lh
        for jj=1:lh
          if (j~=jj)&&mat_conc(h(j),h(jj))==1
              a=[a 1];

          end
        end
    end


    vari=lh*lh-lh;
    if length(a)==vari

     tr=[tr i];
     Texp=[Texp i];
     tt=tt+1;

     P11=find(Post(:,i));

     P1=[P1 P11'];
     if ~isempty(P11)
         P11s(pp)={P11'};
         pp=pp+1;
     end
     for kk=kk+1:length(P11)+kk
         P1ord=[P1ord kk];
     end

     for t=1:length(h)
         Pre_u(h(t),tt)=1;
     end
```

```
                    for kkk=kkk+1:length(P11)+kkk
                        Post_u(P1ord(kkk),tt)=1;
                    end

                    Pexp=[Pexp P11'];

          if ~is_cutoff1(Pre,Post,M0,Pre_u,Post_u,t_tier,Texp)
                       Pexp1=[Pexp1 P11'];
          else
                    [ca nu]=size(Pre_u);
                    cut_off1=[cut_off1 nu];
                    Pexp1=[Pexp1 zeros(1,length(P11))];
          end

                    for bb=length(Pexp_ord)+1:length(Pexp)
                        Pexp_ord=[Pexp_ord bb];
                       end


                       for cc=cc+1:cc+length(P11)
                           Pi_ord=[Pi_ord cc];
                       end

                   end
               end
           end
      end

end
t_tier(g)={tr};
p_tier(g+1)={P1};

mat_conc1=mat_conc;
[mm1 nn1]=size(mat_conc1);

for vo1=1:length(P1ord)

    tra=find(Post_u(P1ord(vo1),:));
    pos=find(Pre_u(:,tra));
    matri=[];
    for op=1:length(pos)
        matri=[matri;mat_conc(pos(op),:)];
```

```
        if op>1
            matri=matri(1,:)&matri(2,:);
        end
    end

    mat_conc2(vo1,:)=matri;

end

mat_conc3=mat_conc2';
mat_conc4=zeros(length(P1ord));

for vvo1=1:length(P1ord)

    ttra=find(Post_u(P1ord(vvo1),:));
    ppos=find(Pre_u(:,ttra));
    mmatri=[];
    for oop=1:length(ppos)
        mmatri=[mmatri;mat_conc3(ppos(oop),:)];
        if oop>1
            mmatri=mmatri(1,:)&mmatri(2,:);
        end
    end

    mat_conc4(vvo1,:)=mmatri;

end

lung1=0;
lung=0;
for mlp=1:length(P11s)
    mom=cell2mat(P11s(mlp));
    lung1=lung1+lung;
    lung=length(mom);

    if lung>1
      matd1=ones(lung);

      for lo=1:lung
          matd1(lo,lo)=0;
      end
      mat_conc4(lung1+1:lung1+lung,lung1+1:lung1+lung)=matd1;
    end
```

```
    end

    mat_conc=[mat_conc1 mat_conc3;mat_conc2 mat_conc4];

    Pi_ord=[];
    Pi_old=P1;
    Pi_old_ord=P1ord;

    P1=[];
    P11=[];
    P1p=[];
    P1ord=[];
    kkk=0;
    g=g+1;
    mat_conc1=[];
    mat_conc2=[];
    mat_conc3=[];
    mat_conc4=[];

end


for ss=1:length(p_tier)
    if isempty(cell2mat(p_tier(ss)))
        p_tier(ss)=[];
    end
end for s=1:length(t_tier)
    if isempty(cell2mat(t_tier(s)))
        t_tier(s)=[];
    end
end if ~isequal(size(Pre_u),size(Post_u))
    [m,n]=size(Pre_u);
    [m1,n1]=size(Post_u);

    for kk=m+1:m1
        Pre_u(kk,:)=0;
    end
end

U={Pre_u,Post_u,p_tier,t_tier,Pexp,Texp,cut_off1,mat_conc};
```

B.3 Funzione order2

```
function[Pre_u,Post_u,p_tier,t_tier,Pexp,Texp,cut_off1,cut_off2]=
=order2(Pre,Post,M0)

M0=M0';
Texp=[];
t_tier(1)={1};
[m,n]=size(Pre);
P0=find(M0);
p_tier(1)={P0};
cc=length(P0);
tt=0;
i=0;
kkk=0;
Pexp=P0;
Pexp1=P0;
Pi_ord=[];
P0ord=[];
P1ord=[];
Pre_u=[];
Post_u=[];
P1=[];
P11=[];
cut_off1=[];
cut_off2=[];

for kk=1:length(P0)
    P0ord=[P0ord kk];
end

Pi_old_ord=P0ord;
Pexp_ord=P0ord;
mat_conc=ones(cc);

for lo=1:cc
    mat_conc(lo,lo)=0;
end g=1;

while ~isempty(cell2mat(p_tier(g)))

    pp=1;
```

```
P11s={};
tr=[];

for i=1:n

    Ptot=[];
    Ptot_ord=Pexp_ord;

            for j=1:length(Pexp_ord)
        for k=1:length(Pi_ord)
            if Pexp_ord(j)==Pi_ord(k)

                Ptot_ord(j)=[0];
            end
        end
    end
    [u, v, Ptot_ord]=find(Ptot_ord);


    for ii=1:length(Ptot_ord)
        in=Ptot_ord(ii);

        Ptot=[Ptot Pexp1(in)];
    end


    stessi={};
    pre_current=Pre(:,i);
    ind=find(pre_current);
    for g12=1:length(ind)
        stes=[];
        for gg12=1:length(Ptot)
            if ind(g12)==Ptot(gg12)
                stes=[stes gg12];
            end
        end
        stessi(g12)={stes};


    end

    la=cell2mat(stessi(1))';
```

```
x=length(la);
for i12=1:length(stessi)-1
    a2=[];
    b=cell2mat(stessi(i12+1));
    for k12=1:x
        for j12=1:length(b)
            a3=[ la(k12,:) b(j12)];
            a2=[a2;a3];
        end
    end
    la=a2;
    [x xx]=size(la);
end
PPsubtot=la;

[mm,nn]=size(PPsubtot);

 for z=1:mm
    h=PPsubtot(z,:);
    if ~isempty(find(h))

        A=[];

        for p=1:length(h),
            for q=1:length(Pi_old_ord)
                if (h(p)==Pi_old_ord(q))
                    A=[A h(p)];
                end
            end
        end

        if ~isempty(A);

            a=[];
            lh=length(h);
            for j=1:lh
                for jj=1:lh
                    if (j~=jj)&&mat_conc(h(j),h(jj))==1
                        a=[a 1];
                    end
                end
            end
            vari=lh*lh-lh;
```

```
            if length(a)==vari
                tr=[tr i];
                Texp=[Texp i];
                tt=tt+1;

                P11=find(Post(:,i));

                P1=[P1 P11'];
                if ~isempty(P11)
                    P11s(pp)={P11'};
                    pp=pp+1;
                end
                for kk=kk+1:length(P11)+kk
                    P1ord=[P1ord kk];
                end

                for t=1:length(h)
                    Pre_u(h(t),tt)=1;
                end

                for kkk=kkk+1:length(P11)+kkk
                    Post_u(P1ord(kkk),tt)=1;
                end

                Pexp=[Pexp P11'];

        if is_cutoff1(Pre,Post,M0,Pre_u,Post_u,t_tier,Texp)

            [ca nu]=size(Pre_u);
            lc=firing_seq(Pre_u,Post_u,t_tier);
            inter=[];

                    for p=1:length(lc),
                        for q=1:length(cut_off1)
                            if (lc(p)==cut_off1(q))
                                inter=[inter lc(p)];
                            end
                        end
                    end
                    if isempty(inter)

                        cut_off1=[cut_off1 nu];
```

```
                        end
                    end

if ~is_cutoff2(Pre,Post,M0,Pre_u, Post_u,Texp,t_tier,cut_off1)

                        Pexp1=[Pexp1 P11'];
                    else

                        [ca nu]=size(Pre_u);
                        cut_off2=[cut_off2 nu];
                        Pexp1=[Pexp1 zeros(1,length(P11))];
                    end

                    for bb=length(Pexp_ord)+1:length(Pexp)
                        Pexp_ord=[Pexp_ord bb];
                    end


                    for cc=cc+1:cc+length(P11)
                        Pi_ord=[Pi_ord cc];
                    end

                end
            end
        end
    end

end

t_tier(g)={tr};
p_tier(g+1)={P1};
mat_conc1=mat_conc;
[mm1 nn1]=size(mat_conc1);

for vo1=1:length(P1ord)

        tra=find(Post_u(P1ord(vo1),:));
        pos=find(Pre_u(:,tra));
        matri=[];
        for op=1:length(pos)
            matri=[matri;mat_conc(pos(op),:)];
            if op>1
```

```
                matri=matri(1,:)&matri(2,:);
            end
        end

        mat_conc2(vo1,:)=matri;
 end

mat_conc3=mat_conc2';
mat_conc4=zeros(length(P1ord));

for vvo1=1:length(P1ord)

        ttra=find(Post_u(P1ord(vvo1),:));
        ppos=find(Pre_u(:,ttra));
        mmatri=[];
        for oop=1:length(ppos)
            mmatri=[mmatri;mat_conc3(ppos(oop),:)];
            if oop>1
                mmatri=mmatri(1,:)&mmatri(2,:);
            end
        end

        mat_conc4(vvo1,:)=mmatri;

end

lung1=0;
lung=0;
for mlp=1:length(P11s)
    mom=cell2mat(P11s(mlp));
    lung1=lung1+lung;
    lung=length(mom);

    if lung>1
      matd1=ones(lung);

      for lo=1:lung
         matd1(lo,lo)=0;
      end
    mat_conc4(lung1+1:lung1+lung,lung1+1:lung1+lung)=matd1;
    end
end
```

```
    mat_conc=[mat_conc1 mat_conc3;mat_conc2 mat_conc4];

    Pi_ord=[];
    Pi_old=P1;
    Pi_old_ord=P1ord;
    P1=[];
    P11=[];
    P1p=[];
    P1ord=[];
    kkk=0;
    g=g+1;
    mat_conc1=[];
    mat_conc2=[];
    mat_conc3=[];
    mat_conc4=[];

end

for ss=1:length(p_tier)
    if isempty(cell2mat(p_tier(ss)))
        p_tier(ss)=[];
    end
end for s=1:length(t_tier)
    if isempty(cell2mat(t_tier(s)))
        t_tier(s)=[];
    end
end if ~isequal(size(Pre_u),size(Post_u))
    [m,n]=size(Pre_u);
    [m1,n1]=size(Post_u);


    for kk=m+1:m1
        Pre_u(kk,:)=0;
    end
end
```

B.4 McMillan

```
function[Pre_u,Post_u,p_tier,t_tier,Pexp,Texp,cut_off,
```

```
cut_off_ord]=McMillan(Pre,Post,M0)

M0=M0';
Texp=[];
t_tier(1)={1.1};
[m,n]=size(Pre);
P0=find(M0);
p_tier(1)={P0};
cc=length(P0);
tt=0;
i=0;
kkk=0;
Pexp=P0;
Pexp1=P0;
Pi_ord=[];
P0ord=[];
P1ord=[];
Pre_u=[];
Post_u=[];
P1=[];
P11=[];
P1p=[];
mm11=M0';
cut_off=[];
cut_off_ord=[];
mat_conc=ones(cc);

for lo=1:cc
    mat_conc(lo,lo)=0;
end

for kk=1:length(P0)
    P0ord=[P0ord kk];
end

Pi_old_ord=P0ord;
Pexp_ord=P0ord;

g=1;

while ~isempty(cell2mat(p_tier(g)))

    pp=1;
```

```
P11s={};
m11=mm11;
P111=[];

tr=[];

for i=1:n

    Ptot=[];
    Ptot_ord=Pexp_ord;

    for j=1:length(Pexp_ord)
        for k=1:length(Pi_ord)
            if Pexp_ord(j)==Pi_ord(k)

                Ptot_ord(j)=[0];
            end
        end
    end
    [u, v, Ptot_ord]=find(Ptot_ord);

    for ii=1:length(Ptot_ord)
        in=Ptot_ord(ii);

        Ptot=[Ptot Pexp1(in)];
    end

    stessi={};
    pre_current=Pre(:,i);
    ind=find(pre_current);
    for g12=1:length(ind)
        stes=[];
        for gg12=1:length(Ptot)
            if ind(g12)==Ptot(gg12)
                stes=[stes gg12];
            end
        end
        stessi(g12)={stes};
    end

    la=cell2mat(stessi(1))';
    x=length(la);
```

```
for i12=1:length(stessi)-1
    a2=[];
    b=cell2mat(stessi(i12+1));
    for k12=1:x
        for j12=1:length(b)
            a3=[ la(k12,:) b(j12)];
            a2=[a2;a3];
        end
    end
    la=a2;
    [x xx]=size(la);
end

PPsubtot=la;

[mm,nn]=size(PPsubtot);

for z=1:mm
    h=PPsubtot(z,:);
    if ~isempty(find(h))

        A=[];

        for p=1:length(h),
            for q=1:length(Pi_old_ord)
                if (h(p)==Pi_old_ord(q))
                    A=[A h(p)];
                end
            end
        end

        if ~isempty(A);

            a=[];
            lh=length(h);
            for j=1:lh
                for jj=1:lh
                  if (j~=jj)&&mat_conc(h(j),h(jj))==1
                      a=[a 1];
                  end
                end
            end
            vari=lh*lh-lh;
```

```
                    if length(a)==vari

                        tr=[tr i];
                        Texp=[Texp i];
                        tt=tt+1;
                        P11=find(Post(:,i));
                        P1=[P1 P11'];

                        if ~isempty(P11)
                            P11s(pp)={P11'};
                            pp=pp+1;
                        end

                        for kk=kk+1:length(P11)+kk
                            P1ord=[P1ord kk];
                        end

                        for t=1:length(h)
                            Pre_u(h(t),tt)=1;
                        end

                        for kkk=kkk+1:length(P11)+kkk
                            Post_u(P1ord(kkk),tt)=1;
                        end

                        Pexp=[Pexp P11'];

                        for bb=length(Pexp_ord)+1:length(Pexp)
                            Pexp_ord=[Pexp_ord bb];
                        end

                        for cc=cc+1:cc+length(P11)
                            Pi_ord=[Pi_ord cc];
                        end
                        P111=[P111 P11'];
                    end
                end
            end
        end
    end

[ctf_aus,ctf_ord_aus,r1,aus,mm11]=
=is_cutoffMcm(Pre_u,Post_u,Texp,Pre,Post,M0,t_tier,m11);
```

```
a=length(Pexp);
b=length(Pexp1);
Pexp11=[Pexp1 Pexp(b+1:a)];

if(isempty(r1))
    Pexp1=[Pexp1 Pexp(b+1:a)];
elseif(~isempty(r1))

    cut_off=[cut_off ctf_aus];
    cut_off_ord=[cut_off_ord ctf_ord_aus];

    for dd=1:length(aus)
        Pexp11(aus(dd))=0;
    end
    Pexp1=Pexp11;
end

mat_conc1=mat_conc;
[mm1 nn1]=size(mat_conc1);

for vo1=1:length(P1ord)

    tra=find(Post_u(P1ord(vo1),:));
    pos=find(Pre_u(:,tra));
    matri=[];

    for op=1:length(pos)
        matri=[matri;mat_conc(pos(op),:)];
        if op>1
            matri=matri(1,:)&matri(2,:);
        end
    end

    mat_conc2(vo1,:)=matri;

end

mat_conc3=mat_conc2';
mat_conc4=zeros(length(P1ord));

for vvo1=1:length(P1ord)
```

```
        ttra=find(Post_u(P1ord(vvo1),:));
        ppos=find(Pre_u(:,ttra));
        mmatri=[];

        for oop=1:length(ppos)
            mmatri=[mmatri;mat_conc3(ppos(oop),:)];
            if oop>1
                mmatri=mmatri(1,:)&mmatri(2,:);
            end
        end

        mat_conc4(vvo1,:)=mmatri;

    end

    lung1=0;
    lung=0;

    for mlp=1:length(P11s)

        mom=cell2mat(P11s(mlp));
        lung1=lung1+lung;
        lung=length(mom);

        if lung>1
         matd1=ones(lung);

         for lo=1:lung
             matd1(lo,lo)=0;
         end
       mat_conc4(lung1+1:lung1+lung,lung1+1:lung1+lung)=matd1;
        end
    end
    mat_conc=[mat_conc1 mat_conc3;mat_conc2 mat_conc4];

    t_tier(g)={tr};
    p_tier(g+1)={P1};

    Pi_ord=[];
    Pi_old=P1;
    Pi_old_ord=P1ord;

    P1=[];
```

```
    P11=[];
    P1p=[];
    P1ord=[];
    kkk=0;
    g=g+1;

    mat_conc1=[];
    mat_conc2=[];
    mat_conc3=[];
    mat_conc4=[];

end

for ss=1:length(p_tier)
    if isempty(cell2mat(p_tier(ss)))
        p_tier(ss)=[];
    end
end

for s=1:length(t_tier)
    if isempty(cell2mat(t_tier(s)))
        t_tier(s)=[];
    end
end

if ~isequal(size(Pre_u),size(Post_u))
    [m,n]=size(Pre_u);
    [m1,n1]=size(Post_u);
    for kk=m+1:m1
        Pre_u(kk,:)=0;
    end
end
```

B.5 Funzione unfolding_graph

```
function unfolding_graph(Pre,Post,M0,f)

[Pre_u,Post_u,p_tier,t_tier,Pexp,Texp]=unfolding_n(Pre,Post,M0,f);
t=0; p=0;
```

```
fprintf('*******************************************\n')

for i=1:length(p_tier)


    fprintf('tier %d:   ',i-1);
    a=cell2mat(p_tier(i));
    for j=1:length(a)
        t=t+1;
        fprintf('P%d',a(j));
        aa=ismember(Pexp(1:t),a(j));
        aa1=length(find(aa));
        if aa1~=1
            fprintf('.%d',aa1-1);
        end
        fprintf('   ')
    end
    fprintf('\n\n')
    if i<=length(t_tier)
        b=cell2mat(t_tier(i));
        for jj=1:length(b)
            p=p+1;
            fprintf('t%d',b(jj));
               cc=ismember(Texp(1:p),b(jj));
        cc1=length(find(cc));
        if cc1~=1
            fprintf('.%d',cc1-1);
        end
        fprintf('   ')

             fprintf('\n          in:  ')
            pr=find(Pre_u(:,p));
            for k=1:length(pr)
                bb=Pexp(pr(k));
                fprintf('P%d',bb);
                mm=ismember(Pexp(1:pr(k)),bb);
                mm1=length(find(mm));
                if mm1~=1
                    fprintf('.%d',mm1-1);
                end
                fprintf('   ')
            end
            fprintf('\n         out:  ')
```

```
            po=find(Post_u(:,p));
            fprintf('')
            for kk=1:length(po)
                bb1=Pexp(po(kk));
                fprintf('P%d',bb1)
                nn=ismember(Pexp(1:po(kk)),bb1);
                nn1=length(find(nn));
                   if nn1~=1
                    fprintf('.%d',nn1-1);
                end
                fprintf('   ')
            end

            fprintf('\n\n')
        end

        fprintf('\n')
    end
    fprintf('*********************************************\n')
    pause
end
```

B.6 Funzione graph1

```
function graph1(Pre,Post,M0)

[Pre_u,Post_u,p_tier,t_tier,Pexp,Texp,cut_off1,U]=
=order1(Pre,Post,M0);

t=0; p=0;
fprintf('*************************************************\n')
 for i=1:length(p_tier)

    fprintf('tier %d:   ',i-1);
    a=cell2mat(p_tier(i));
    for j=1:length(a)
        t=t+1;
        fprintf('P%d',a(j));
        aa=ismember(Pexp(1:t),a(j));
        aa1=length(find(aa));
```

```
        if aa1~=1
            fprintf('.%d',aa1-1);
        end
        fprintf('   ')
    end
    fprintf('\n\n')
    if i<=length(t_tier)
        b=cell2mat(t_tier(i));
        for jj=1:length(b)
            p=p+1;
            fprintf('t%d',b(jj));
               cc=ismember(Texp(1:p),b(jj));
        cc1=length(find(cc));
        if cc1~=1
            fprintf('.%d',cc1-1);
        end
        fprintf('   ')

             fprintf('\n          in:  ')
            pr=find(Pre_u(:,p));
            for k=1:length(pr)
                bb=Pexp(pr(k));
                fprintf('P%d',bb);
                mm=ismember(Pexp(1:pr(k)),bb);
                mm1=length(find(mm));
                if mm1~=1
                    fprintf('.%d',mm1-1);
                end
                fprintf('   ')
            end
            fprintf('\n         out:  ')
            po=find(Post_u(:,p));
            fprintf('')
            for kk=1:length(po)
                bb1=Pexp(po(kk));
                fprintf('P%d',bb1)
                nn=ismember(Pexp(1:po(kk)),bb1);
                nn1=length(find(nn));
                   if nn1~=1
                    fprintf('.%d',nn1-1);
                end
                fprintf('   ')
            end
```

```
            fprintf('\n\n')
        end



        fprintf('\n')
    end
    fprintf('*******************************************\n')
    pause
end

fprintf('cut_off: ')

        for kkk=1:length(cut_off)
                bb11=Texp(cut_off(kkk));
                fprintf('t%d',bb11)
                vv=ismember(Texp(1:cut_off(kkk)),bb11);
                vv1=length(find(vv));
                   if vv1~=1
                    fprintf('.%d',vv1-1);
                end
                fprintf(' ')
            end
```

B.7 Funzione graph2

```
function graph2(Pre,Post,M0)

[Pre_u,Post_u,p_tier,t_tier,Pexp,Texp,cut_off1,cut_off2]=
=order2(Pre,Post,M0); t=0; p=0;

fprintf('*********************************************\n')

 for i=1:length(p_tier)

    fprintf('TIER %d:   ',i-1);
    a=cell2mat(p_tier(i));
    for j=1:length(a)
        t=t+1;
```

```
        fprintf('P%d',a(j));
        aa=ismember(Pexp(1:t),a(j));
        aa1=length(find(aa));
        if aa1~=1
            fprintf('.%d',aa1-1);
        end
        fprintf('   ')
    end
    fprintf('\n\n')
    if i<=length(t_tier)
        b=cell2mat(t_tier(i));
        for jj=1:length(b)
            p=p+1;
            fprintf('t%d',b(jj));
               cc=ismember(Texp(1:p),b(jj));
        cc1=length(find(cc));
        if cc1~=1
            fprintf('.%d',cc1-1);
        end
        fprintf('   ')

             fprintf('\n          in:  ')
            pr=find(Pre_u(:,p));
            for k=1:length(pr)
                bb=Pexp(pr(k));
                fprintf('P%d',bb);
                mm=ismember(Pexp(1:pr(k)),bb);
                mm1=length(find(mm));
                if mm1~=1
                    fprintf('.%d',mm1-1);
                end
                fprintf('   ')
            end
            fprintf('\n         out:  ')
            po=find(Post_u(:,p));
            fprintf('')
            for kk=1:length(po)
                bb1=Pexp(po(kk));
                fprintf('P%d',bb1)
                nn=ismember(Pexp(1:po(kk)),bb1);
                nn1=length(find(nn));
                   if nn1~=1
                    fprintf('.%d',nn1-1);
```

```
                end
                fprintf('   ')
            end

            fprintf('\n\n')
        end



        fprintf('\n')
    end
    fprintf('*****************************************\n')
    pause
end fprintf('cut_off1: ')


        for kkk=1:length(cut_off1)
                bb11=Texp(cut_off1(kkk));
                fprintf('t%d',bb11)
                vv=ismember(Texp(1:cut_off1(kkk)),bb11);
                vv1=length(find(vv));
                   if vv1~=1
                    fprintf('.%d',vv1-1);
                end
                fprintf(' ')
            end
            fprintf('\n\n')
            fprintf('cut_off2: ')


        for k2=1:length(cut_off2)
                b2=Texp(cut_off2(k2));
                fprintf('t%d',b2)
                v2=ismember(Texp(1:cut_off2(k2)),b2);
                vv2=length(find(v2));
                   if vv2~=1
                    fprintf('.%d',vv2-1);
                end
                fprintf(' ')
            end
```

B.8 Funzione graphMcM

```
function graphMcM(Pre,Post,M0)

[Pre_u,Post_u,p_tier,t_tier,Pexp,Texp,cut_off,cut_off_ord]=
=McMillan(Pre,Post,M0); t=0; p=0;

fprintf('************************************************\n')

for i=1:length(p_tier)

    fprintf('tier %d:   ',i-1);
    a=cell2mat(p_tier(i));
    for j=1:length(a)
        t=t+1;
        fprintf('P%d',a(j));
        aa=ismember(Pexp(1:t),a(j));
        aa1=length(find(aa));
        if aa1~=1
            fprintf('.%d',aa1-1);
        end
        fprintf('   ')
    end
    fprintf('\n\n')
    if i<=length(t_tier)
        b=cell2mat(t_tier(i));
        for jj=1:length(b)
            p=p+1;
            fprintf('t%d',b(jj));
            cc=ismember(Texp(1:p),b(jj));
            cc1=length(find(cc));
            if cc1~=1
                fprintf('.%d',cc1-1);
            end
            fprintf('   ')

            fprintf('\n          in:  ')
            pr=find(Pre_u(:,p));
            for k=1:length(pr)
                bb=Pexp(pr(k));
                fprintf('P%d',bb);
                mm=ismember(Pexp(1:pr(k)),bb);
```

```
                mm1=length(find(mm));
                if mm1~=1
                    fprintf('.%d',mm1-1);
                end
                fprintf('   ')
            end
            fprintf('\n         out:  ')
            po=find(Post_u(:,p));
            fprintf('')
            for kk=1:length(po)
                bb1=Pexp(po(kk));
                fprintf('P%d',bb1)
                nn=ismember(Pexp(1:po(kk)),bb1);
                nn1=length(find(nn));
                if nn1~=1
                    fprintf('.%d',nn1-1);
                end
                fprintf('   ')
            end

            fprintf('\n\n')
        end

        fprintf('\n')
    end
    fprintf('*****************************************\n')
    pause
end

fprintf('cut_off: ')

for kkk=1:length(cut_off_ord)
    bb11=Texp(cut_off_ord(kkk));
    fprintf('t%d',bb11)
    vv=ismember(Texp(1:cut_off_ord(kkk)),bb11);
    vv1=length(find(vv));
    if vv1~=1
        fprintf('.%d',vv1-1);
    end
    fprintf(' ')
end
```

B.9 Funzione reach_u

```
function [ret,p_corr,X]=reach_u(M0,M,U,opt)


na=nargin;
t_tier=U{4};
switch na
    case 3
        ret=0;
        Pexp=cell2mat(U(5));
        mat_conc=cell2mat(U(8));

        if M==M0
            ret=1;
            return
        end
        pm=find(M);
        pnm=find(M==0);
        lpm=length(pm);
        for g=1:length(pm)
            stes=[];
            for gg=1:length(Pexp)
                if pm(g)==Pexp(gg)
                    stes=[stes gg];
                end
            end
            stessi(g)={stes};

        end

        inter=ismember(Pexp,pm);
        [aa posti_ord c]=find(inter);

        inter1=ismember(Pexp,pnm);
        [aa1 nposti_ord c1]=find(inter1);



        la=cell2mat(stessi(1))';

        x=length(la);
```

```
for i=1:length(stessi)-1
    a2=[];
    b=cell2mat(stessi(i+1));
    for k=1:x
        for j=1:length(b)
            a3=[ la(k,:) b(j)];
            a2=[a2;a3];
        end
    end
    la=a2;
    [x xx]=size(la);
end
da_controllare=la;


[uno due]=size(da_controllare);
for t=1:uno
    ex=da_controllare(t,:);

    a=[];
    lex=length(ex);
    for j=1:lex
        for jj=1:lex
            if (j~=jj)&&mat_conc(ex(j),ex(jj))==1
                a=[a 1];
            end
        end
    end

    vari=lex*lex-lex;

    if length(a)==vari

        d=[];
        for  tt=1:length(nposti_ord)
            mm=[];
            for t=1:lex

                if mat_conc(ex(t),nposti_ord(tt))==1
                    mm=[mm 1];
                end

            end
```

```
                if length(mm)~=lex


                    d=[d 1];

                end
            end
            if length(d)==length(nposti_ord)
                ret=1;
                return

           end
        end
    end
case 4
    switch opt
        case 1

            ret=0;
            p_corr=[];
            Pexp=cell2mat(U(5));
            mat_conc=cell2mat(U(8));

            pm=find(M);
            pnm=find(M==0);
            lpm=length(pm);
            for g=1:length(pm)
                stes=[];
                for gg=1:length(Pexp)
                    if pm(g)==Pexp(gg)
                        stes=[stes gg];
                    end
                end
                stessi(g)={stes};

            end

            inter=ismember(Pexp,pm);
            [aa posti_ord c]=find(inter);



            inter1=ismember(Pexp,pnm);
```

```
[aa1 nposti_ord c1]=find(inter1);



la=cell2mat(stessi(1))';

x=length(la);
for i=1:length(stessi)-1

    a2=[];
    b=cell2mat(stessi(i+1));
    for k=1:x

        for j=1:length(b)

            a3=[ la(k,:) b(j)];
            a2=[a2;a3];
        end
    end
    la=a2;
    [x xx]=size(la);
end
da_controllare=la;

[uno due]=size(da_controllare);
for t=1:uno
    ex=da_controllare(t,:);

    a=[];
    lex=length(ex);
    for j=1:lex
        for jj=1:lex
         if (j~=jj)&&mat_conc(ex(j),ex(jj))==1
             a=[a 1];
         end
        end
    end

    vari=lex*lex-lex;

    if length(a)==vari

        d=[];
```

```
            for  tt=1:length(nposti_ord)
                mm=[];
              for t=1:lex

              if mat_conc(ex(t),nposti_ord(tt))==1
                 mm=[mm 1];
              end

                end
                if length(mm)~=lex


                    d=[d 1];

                end

            end
            if length(d)==length(nposti_ord)

                p_corr=[p_corr;ex];

            end

        end
    end

if ~isempty(p_corr)
    ret=1;
end
[s1 s2]=size(p_corr);
fprintf('\n')
for r=1:s1
    q=p_corr(r,:);
    fprintf('%d^ set of places:  ',r)
    for rr=1:length(q)
        num=Pexp(q(rr));
        fprintf('p%d',num)
        Pexp1=Pexp(1:q(rr)-1);
        ma=ismember(Pexp1,num);
        ma1=length(find(ma));
        if ma1~=0
            fprintf('.%d  ',ma1)
        else
```

```
                fprintf('   ')
            end
        end
        fprintf('\n')
    end




case 2

    ret=0;
    p_corr=[];
    Pexp=cell2mat(U(5));
    mat_conc=cell2mat(U(8));

    pm=find(M);
    pnm=find(M==0);
    lpm=length(pm);
    for g=1:length(pm)
        stes=[];
        for gg=1:length(Pexp)
            if pm(g)==Pexp(gg)
                stes=[stes gg];
            end
        end
        stessi(g)={stes};


    end


    inter=ismember(Pexp,pm);
    [aa posti_ord c]=find(inter);



    inter1=ismember(Pexp,pnm);
    [aa1 nposti_ord c1]=find(inter1);


    la=cell2mat(stessi(1))';
```

```
x=length(la);
for i=1:length(stessi)-1

    a2=[];
    b=cell2mat(stessi(i+1));
    for k=1:x

        for j=1:length(b)

            a3=[ la(k,:) b(j)];
            a2=[a2;a3];
        end
    end
    la=a2;
    [x xx]=size(la);
end
da_controllare=la;




[uno due]=size(da_controllare);
for t=1:uno
    ex=da_controllare(t,:);

    a=[];
    lex=length(ex);
    for j=1:lex
        for jj=1:lex
         if (j~=jj)&&mat_conc(ex(j),ex(jj))==1
             a=[a 1];
         end
        end
    end

    vari=lex*lex-lex;

    if length(a)==vari

        d=[];
        for  tt=1:length(nposti_ord)
          mm=[];
          for t=1:lex
```

```
            if mat_conc(ex(t),nposti_ord(tt))==1
                  mm=[mm 1];
            end

             end
             if length(mm)~=lex


                   d=[d 1];

             end

           end
           if length(d)==length(nposti_ord)

               p_corr=[p_corr;ex];

           end

       end
 end

 if ~isempty(p_corr)
     ret=1;
 end
 Pre_u=cell2mat(U(1));
 Post_u=cell2mat(U(2));

 Texp=cell2mat(U(6));

 MM=[];

 if ret==0
fprintf('ERROR:the marking is not reachable!!!');
return
 end
 tra1=[];
 a2=[];


 for j=1:length(t_tier)
     a=cell2mat(t_tier(j));
```

```
    a1=length(a);
    a2=[a2 a1];
end


[uni dui]=size(p_corr);
for index=1:uni
    lc_tier={};
    p_corr1=p_corr(index,:);
    for i=1:length(p_corr1)
        tra=find(Post_u(p_corr1(i),:));
        if ~isempty(tra)
            Pre_u1=Pre_u(:,1:tra);
            Post_u1=Post_u(:,1:tra);
            aa=0;
            x=1;
            for t=1:length(a2)
                aa=aa+a2(t);
                if aa<tra
                    x=t+1;
                end
            end

  t_tier1=t_tier(1:x);
lc=firing_seq(Pre_u1,Post_u1,t_tier1);
            for p=1:length(lc)
                bb=0;
                xx=1;
                for tt=1:length(a2)
                    bb=bb+a2(tt);
                    if bb<lc(p)
                        xx=tt+1;
                    end
                end


                if length(lc_tier)~=0
                 if length(lc_tier)>=xx
                    lcpr=cell2mat(lc_tier(xx));
                    lcpr=[lcpr lc(p)];

                 else
                        lcpr=lc(p);
```

```
                end
                    lc_tier(xx)={lcpr};
                else
                    lc_tier(xx)={lc(p)};
                end

            end
        end
    end
    mom11=[];
    for n=1:length(lc_tier)
        mom=cell2mat(lc_tier(n));
        mom=unique(mom);
        mom11=[mom11 mom];
    end
    da_confrontare(index)={mom11};
end


s=[];
for in=1:length(da_confrontare)
    fre=cell2mat(da_confrontare(in));
    s1=length(fre);
    s=[s s1];
end
rest=min(s);
res1=find(s==rest);
mom1=cell2mat(da_confrontare(res1));

X=mom1;
fprintf('\n')
fprintf(' the shortest firing sequence is:  ')
for rr=1:length(X)
    num=Texp(X(rr));
    fprintf('t%d',num)
    Texp1=Texp(1:X(rr)-1);
    ma=ismember(Texp1,num);
    ma1=length(find(ma));
    if ma1~=0
        fprintf('.%d  ',ma1)
    else
        fprintf('   ')
```

```
        end
    end
    fprintf('\n')



case 3
    ret=0;
    p_corr=[];
    Pexp=cell2mat(U(5));
    mat_conc=cell2mat(U(8));

    pm=find(M);
    pnm=find(M==0);
    lpm=length(pm);
    for g=1:length(pm)
        stes=[];
        for gg=1:length(Pexp)
            if pm(g)==Pexp(gg)
                stes=[stes gg];
            end
        end
        stessi(g)={stes};


    end

    inter=ismember(Pexp,pm);
    [aa posti_ord c]=find(inter);



    inter1=ismember(Pexp,pnm);
    [aa1 nposti_ord c1]=find(inter1);




    la=cell2mat(stessi(1))';

    x=length(la);
    for i=1:length(stessi)-1
```

```
    a2=[];
    b=cell2mat(stessi(i+1));
    for k=1:x

        for j=1:length(b)

            a3=[ la(k,:) b(j)];
            a2=[a2;a3];
        end
    end
    la=a2;
    [x xx]=size(la);
end
da_controllare=la;



[uno due]=size(da_controllare);
for t=1:uno
    ex=da_controllare(t,:);

    a=[];
    lex=length(ex);
    for j=1:lex
        for jj=1:lex
         if (j~=jj)&&mat_conc(ex(j),ex(jj))==1
             a=[a 1];
         end
        end
    end

    vari=lex*lex-lex;

    if length(a)==vari

        d=[];
        for  tt=1:length(nposti_ord)
          mm=[];
          for t=1:lex

          if mat_conc(ex(t),nposti_ord(tt))==1
               mm=[mm 1];
           end
```

```
            end
          if length(mm)~=lex


              d=[d 1];

          end

          end
          if length(d)==length(nposti_ord)

              p_corr=[p_corr;ex];

          end

      end
  end

  if ~isempty(p_corr)
      ret=1;
  end
  Pre_u=cell2mat(U(1));
  Post_u=cell2mat(U(2));

  Texp=cell2mat(U(6));

  MM=[];

  if ret==0
fprintf('ERROR:the marking is not reachable!!!');
  return
  end
  tra1=[];
  a2=[];


  for j=1:length(t_tier)
      a=cell2mat(t_tier(j));
      a1=length(a);
      a2=[a2 a1];
  end
```

```
[uni dui]=size(p_corr);
for index=1:uni
    lc_tier={};
    p_corr1=p_corr(index,:);
    for i=1:length(p_corr1)
        tra=find(Post_u(p_corr1(i),:));
        if ~isempty(tra)
            Pre_u1=Pre_u(:,1:tra);
            Post_u1=Post_u(:,1:tra);
            aa=0;
            x=1;
            for t=1:length(a2)
                aa=aa+a2(t);
                if aa<tra
                    x=t+1;
                end
            end

            t_tier1=t_tier(1:x);
lc=firing_seq(Pre_u1,Post_u1,t_tier1);
            for p=1:length(lc)
                bb=0;
                xx=1;
                for tt=1:length(a2)
                    bb=bb+a2(tt);
                    if bb<lc(p)
                        xx=tt+1;
                    end
                end

                if length(lc_tier)~=0
                    if length(lc_tier)>=xx
                   lcpr=cell2mat(lc_tier(xx));
                   lcpr=[lcpr lc(p)];

                    else
                        lcpr=lc(p);
                    end
                    lc_tier(xx)={lcpr};
                else
                    lc_tier(xx)={lc(p)};
                end
            end
```

```
        end
    end
    mom11=[];
    for n=1:length(lc_tier)
        mom=cell2mat(lc_tier(n));
        mom=unique(mom);
        mom11=[mom11 mom];
    end
    da_confrontare(index)={mom11};
end
fprintf('\n')
[s1 s2]=size(p_corr);
for r=1:s1
    q=p_corr(r,:);
    fprintf('%d^ set of places:  ',r)
    for rr=1:length(q)
        num=Pexp(q(rr));
        fprintf('p%d',num)
        Pexp1=Pexp(1:q(rr)-1);
        ma=ismember(Pexp1,num);
        ma1=length(find(ma));
        if ma1~=0
            fprintf('.%d  ',ma1)
        else
            fprintf('   ')
        end
    end
    fprintf('\n')
end

X=da_confrontare;
fprintf('\n')
for r=1:length(X)
    XX=cell2mat(X(r));

    fprintf(' the %d firing sequence is:  ',r)
    for rr=1:length(XX)
        num=Texp(XX(rr));
        fprintf('t%d',num)
        Texp1=Texp(1:XX(rr)-1);
        ma=ismember(Texp1,num);
        ma1=length(find(ma));
        if ma1~=0
```

```
                            fprintf('.%d  ',ma1)
                        else
                            fprintf('   ')
                        end
                    end
                    fprintf('\n')
                end
            otherwise
                fprintf('ERROR:the value of p is uncorrect.
                 You must choice a value between 1 and 3')
        end

      otherwise
        fprintf('ERROR:the number of output is uncorrect')
end
```

B.10 Funzione Filosofi

```
function [Pre,Post,M0]=Filosofi(n,m)

m=m+1;

if (n<2 && m<4)
 fprintf('ERROR! The philosophers have to be more than two and
 the states of every philosopher they have to be at least four!!!')

    Pre=[];
    Post=[];
    M0=[];
    return
end


if (n<2)
 fprintf('ERROR! The philosophers have to be more than two!!!')
    Pre=[];
    Post=[];
    M0=[];
    return
```

```
end if (m<4)
 fprintf('ERROR! The states of every philosopher they
    have to be at least four!!!')

    Pre=[];
    Post=[];
    M0=[];
    return
end


a=[1 1];
for i=1:m-2
    bb(i)=[0];
end mm=cat(2,a,bb);
M0=[];
for i=1:n
    M0=[M0 mm];
end


Pre=zeros(m*n,(m-1)*n);
Post=zeros(m*n,(m-1)*n);

Pre11=zeros(m,m-1);

for i=2:m
    Pre11(i,i-1)=1;
end Pre11(1,1)=1;


j=1; t=1; for i=1:n
    Pre(j:j+m-1,t:t+(m-1)-1)=Pre11;
    j=j+m;
    t=t+(m-1);
end


a=(m-1)+(m-2);
Pre(2,a)=1;
c=2+m;
d=a+(m-1);
for ii=1:n-2
```

```
    Pre(c,d)=1;
    c=c+m;
    d=d+(m-1);
end
Pre(m*n-(m-2),m-2)=1;

Post11=zeros(m,m-1);

for i=3:m

    Post11(i,i-2)=1;
end
Post11(1:2,m-1)=1;


jj=1;
tt=1;
for i=1:n
    Post(jj:jj+m-1,tt:tt+(m-1)-1)=Post11;
    jj=jj+m;
    tt=tt+(m-1);
end


Post(m*n-(m-2),m-1)=1;
aa=(m-1)+(m-1);
Post(2,aa)=1;

cc=2+m;
dd=aa+(m-1);
for i1=1:n-2
    Post(cc,dd)=1;
    cc=cc+m;
    dd=dd+(m-1);
end
M0=M0';
```

B.11 Funzioni ausiliarie

B.11.1 Funzione is_cutoff1

```
function [r]=is_cutoff1(Pre,Post,M0,Pre_u,Post_u,t_tier,Texp)

M0=M0';
r=0;

if isempty(Pre_u)
    return
end

[m1,n1]=size(Post_u);

if ~isequal(size(Pre_u),size(Post_u))
    [m,n]=size(Pre_u);

    for kk=m+1:m1
        Pre_u(kk,:)=0;
    end
end

[lc]=firing_seq(Pre_u,Post_u,t_tier);

for i=1:length(lc)-1
    Pre_u1=Pre_u(:,1:lc(i));
    Post_u1=Post_u(:,1:lc(i));

    [lc1]=firing_seq(Pre_u1,Post_u1,t_tier);

    lc_tier(i)={lc1};
end

 lc_tier(i+1)={lc};
 M=M0;
 for d=1:length(lc_tier)
    M_old=M0;
    lcs=cell2mat(lc_tier(d));

    for f=1:length(lcs)
```

```
        t=Texp(lcs(f));
         M_old=M_old-Pre(:,t)+Post(:,t);

    end
    M=[M M_old];
end

[p q]=size(M);

for h=1:q
    for hh=1:q
        if (h~=hh)&&(isequal(M(:,h),M(:,hh)))
            r=1;
        end
    end
end
```

B.11.2 Funzione is_cutoff2

```
function
[r]=is_cutoff2(Pre,Post,M0,Pre_u,Post_u,Texp,t_tier,cut_off1)

r=0;
if isempty(Pre_u)
    return
end

[m1,n1]=size(Post_u);

 if ~isequal(size(Pre_u),size(Post_u))
    [m,n]=size(Pre_u);



    for kk=m+1:m1
        Pre_u(kk,:)=0;
    end
end

[lc]=firing_seq(Pre_u,Post_u,t_tier);
```

```
lc_real=[];
 for t=1:length(lc)
    lc_real=[lc_real Texp(lc(t))];
end

M_old=M0';

for i=1:length(lc_real)
    M=M_old-Pre(:,lc_real(i))+Post(:,lc_real(i));
    M_old=M;
end o1=[];


for g=1:length(cut_off1)
    Pre_u2=Pre_u(:,1:cut_off1(g));
    Post_u2=Post_u(:,1:cut_off1(g));
    loc=firing_seq(Pre_u2,Post_u2,t_tier);
    o1=[o1 loc];
end


forbidden=setdiff(o1,cut_off1);
if ~isempty(forbidden)
    for j=1:length(lc)-1

        if isempty(find(ismember(forbidden,lc(j))))
            Pre_u1=Pre_u(:,1:lc(j));
            Post_u1=Post_u(:,1:lc(j));
            [lc1]=firing_seq(Pre_u1,Post_u1,t_tier);

            lc_real1=[];
            for jj=1:length(lc1)
                lc_real1=[lc_real1 Texp(lc1(jj))];
            end
            MM_old=M0';
            for ii=1:length(lc_real1)
            MM=MM_old-Pre(:,lc_real1(ii))+Post(:,lc_real1(ii));
            MM_old=MM;
            end

            if MM==M
                mirror=lc(j);
                r=1;
```

```
                return
            end
        end
    end

end
```

B.11.3 Funzione firing_seq

```
function [lc]=firing_seq(Pre_u,Post_u,t_tier)

[m n]=size(Pre_u);

posti=find(Post_u(:,n)); tran=[];
[p]=previous(Pre_u,Post_u,posti(1),t_tier);

p=[p posti(1)];

for i=1:length(p)
    t=find(Post_u(p(i),:));
    tran=[tran t];
end

lc=unique(tran);
```

B.11.4 Funzione is_cutoffMcm

```
function[cut_off,cut_off_ord,r1,aus,MM11]=
=is_cutoffMcm(pre,post,Texp,Pre,Post,M0,t_tier,M11)

aus=[];
cut_off=[];
cut_off_ord=[];
MM11=[];
r1=[];
```

```
if isempty(pre)
    return
end

[m1,n1]=size(post);

if ~isequal(size(pre),size(post))

    [m,n]=size(pre);

    for kk=m+1:m1
        pre(kk,:)=0;
    end
end

[LC]=lc_McMillan(pre,post,t_tier); Mc1=[] ;


for j=1:length(LC)

    LC1=LC{j};
    lc_real=[];

    for t=1:length(LC1)

        lc_real=[lc_real Texp(LC1(t))];

    end

    M_old=M0';

    for tt=1:length(lc_real)
        M=M_old-Pre(:,lc_real(tt))+Post(:,lc_real(tt));
        M1=[M];
        M_old=M;
    end

    [p1,q1]=size(M11);
    [p q]=size(M1);

    for h=1:q
        for hh=1:q1
```

```
            if (isequal(M1(:,h),M11(:,hh)))

                cut_off=[cut_off lc_real(end)];
                cut_off_ord=[cut_off_ord LC1(end)];

                LC1=LC{j};
                [a,v,m]=find(post(:,LC1(end)));
                r=1;
                r1=[r1 r ];
                aus=[aus; a];

                break
            else
                a=0;
            end
        end
    end

    Mc1=[Mc1 M1];

end

MM1=[M11 Mc1];
MM11=(unique(MM1',''rows''))';
```

B.11.5 Funzione lc_McMillan

```
function [LC]=lc_McMillan(pre,post,t_tier)

if cell2mat(t_tier)==1.1
    d=0;
else
    d=length(cell2mat(t_tier));
end [m,n]=size(pre); LC=[]; lc1=[]; ii=1; for j=d+1:n

    posti=find(post(:,j));
    tran=[];
    [p]=previous(pre,post,posti(1),t_tier);
    p=[p posti(1)];
```

```
    for i=1:length(p)
        t=find(post(p(i),:));
        tran=[tran t];
    end
    tran=sort(tran);

    if(~isempty(tran))
        lc=tran(1);
        for i=1:length(tran)-1

            if(tran(i)~=tran(i+1))
                lc=[lc tran(i+1)];

            end
        end
    else
        lc=tran;
    end

    LC{ii}=lc;
    ii=ii+1;

end
```

B.11.6 Funzione previous

```
function [posti3]=previous(Pre,Post,posti,t_tier)

posti2=[]; for r=1:length(t_tier)
  posti1=[];
    for i=1:length(posti)
        tran=find(Post(posti(i),:));
        posti_new=find(Pre(:,tran));
        posti1=[posti1 posti_new'];
         posti2=[posti2 posti_new'];
    end
    posti=posti1;
end posti3=unique(posti2);
```

Bibliografia

- A. Benveniste, E. Fabre, S. Haar e C. Jard (2003), *Diagnosis of asynchronous discrete event systems, a net unfolding approach*, IEEE Trans. on Automatic Control 48(5), 714-727.
- A. Di Febbraro, A. Giua (2003), *Sistemi ad eventi Discreti*, McGraw-Hill.
- J. Esparza, S. Römer e W. Vogler (2002), *An improvement of McMillan's unfolding algorithm*, Formal Methods in System Design 20, 285-310.
- A. Giua, X. Xie (2005), *Control of safe ordinary Petri nets using unfolding*.
- A. Giua, X. Xie (2004), *Counterexamples to "Liveness-Enforcing Supervision of Bounded Ordinary Petri Nets Using Partial-Order Methods"*, IEEE TRANSACTIONS ON AUTOMATIC CONTROL, VOL. 49, NO. 7.
- K. X. He, M. D. Lemmon, (2002). *Liveness-enforcing supervision of bounded ordinary Petri nets using partial order methods*. IEEE Trans. on Automatic Control 47(7), 1042-1055.
- K.L. McMillan (1995), *A technique of state space search based on unfolding*. Formal Methods in System Design 6(1), 45-65.