UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

ELABORATO DI LAUREA IN

INGEGNERIA DELL'AUTOMAZIONE

Non-interferenza mediante controllo supervisivo in reti di Petri limitate

Relatore
Chiar.mo Prof.
Gianmaria De Tommasi

Candidato
Ciro Mazzocchi
N39/456

Anno Accademico 2017/2018

"I FALLIMENTI NON SONO UN QUALCOSA DA EVITARE, ANZI, È MEGLIO FALLIRE IL PRIMA POSSIBILE COSÌ DA POTER MIGLIORARE PIÙ IN FRETTA."

GORDON MOORE

1 Indice

2 Introduzione

Alla base della teoria dei sistemi vi è il conetto che qualsiasi sistema può essere analizzato attraverso delle equazioni che descrivono l'evoluzione del sistema in funzione degli ingressi e delle variabili che indicano lo stato del sistema nell'istante t.

Al crescere del sistema aumenta il numero di funzioni ed il numero di variabili di stato, rendendo praticamente impossibile proseguire su questa strada.

In realtà in grossi sistemi industriali non abbiamo neanche bisogno di un tale livello di precisione, ma possiamo analizzare l'evoluzione del sistema attraverso un insieme di eventi d'ingresso e un insieme di stati discreto. Un sistema così descritto viene definito *Sistema ad Eventi Discreti* (SED).

Nel 1962, nella sua tesi di dottorato *Kommunikation mit Automaten*, Adam Carl Petri presenta un nuovo modello matematico per descrivere i SED, le reti di Petri.

Queste, attraverso un sistema di token e di transizioni, possono facilmente descrivere l'evoluzione nel tempo di un sistema SED.

Attraverso l'utilizzo di reti di alto livello e basso livello, le reti di Petri possono descrivere e verificare la sicurezza di un sistema, verificando che un intruso (un utente di basso livello) non carpisca informazioni sul funzionamento della rete di alto livello, ossia non vi sia una perdita di informazioni.

Di seguito descritta la struttura della tesi.

Il capitolo 3, diviso in due parti, è dedicato alla descrizione del modello matematico e dei teoremi alla base dell'algoritmo implementato.

Nella prima parte del capitolo viene introdotto il modello matematico alla base della rete di Petri, indicando come viene definito lo stato della rete e come la rete evolve nel tempo.

Nella seconda parte del capitolo vengono introdotte le definizioni di rete di Petri di alto livello e basso livello, viene definita la proprietà di *Strong Non deterministic Non Interference* (SNNI) ed il teorema che caratterizza una rete SNNI.

Il capitolo 4, diviso in tre parti, contiene il cuore della tesi in quanto vengono descritte le specifiche del problema e l'implementazione dell'algoritmo.

Nella prima parte del capitolo viene riportato lo pseudo-codice descritto in [2], spiegando riga per riga il suo funzionamento.

Nella seconda parte del capitolo vengono descritti gli strumenti utilizzati nell'implementazione della tesi.

La terza parte del capitolo viene descritto riga per riga l'algoritmo implementato, motivando ogni scelta progettuale. Inizialmente vengono descritte il main e le funzioni principali dell'algoritmo, successivamente vengono descritte le funzioni di supporto all'algoritmo. Inoltre, viene descritta la funzione test_SNNI.m che, anche se non fa parte

dell’algoritmo finale, sarà utile durante gli esempi del capitolo 5 per verificare che la rete sia SNNI.

Nel capitolo 5 vengono riportati due casi d’uso dell’algoritmo, evidenziando in particolare come una scelta sbagliata dei valori di input potrebbe rendere il problema non risolvibile oppure sicuramente risolvibile, ma dal risultato inutile.

Nel capitolo 6, infine, vengono riportate le conclusioni evidenziando eventuali sviluppi futuri del progetto.

3 Richiami sulle reti di Petri

In questo capitolo saranno riportate le definizioni e i teoremi alla base del codice implementato.

La prima parte del capitolo fornirà la base matematica necessaria per la comprensione dell'algoritmo, descrivendo inizialmente il modello matematico alla base della rete di Petri, successivamente la funzione di marcatura utile per definire lo stato della rete, ed infine il vettore di scatto per spiegare la dinamica di una rete.

Nella seconda parte del capitolo saranno riportate le definizioni di rete di alto livello e basso livello, successivamente verrà data una definizione di proprietà SNNI ed infine verrà riportato il teorema che descrivere una rete SNNI.

3.1 Reti di Petri

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

Formalmente, la definizione di rete di Petri è al seguente:

Definizione 3.1: *Una rete di Petri è definita da quadrupla*

$$N = (P, T, Pre, Post) \quad [3.1]$$

dove:

- $P = \{p_1, p_2, \dots, p_m\}$ *è l'insieme degli m posti.*
- $T = \{t_1, t_2, \dots, t_n\}$ *è l'insieme delle n transizioni*
- $Pre: P\ x\ T \rightarrow \mathbb{N}$ *è la funzione di pre-incidenza che specifica gli archi diretti dai posti alle transizioni e viene rappresentata mediante una matrice di interi non negativi m x n. Più esattamente, indica quanti archi vanno dal posto p alla transizione t.*
- $Post: P\ x\ T \rightarrow \mathbb{N}$ *è la funzione di post-incidenza che specifica gli archi diretti dalle transizioni ai posti e viene rappresentata mediante una matrice di interi non negativi m x n. Più esattamente, indica quanti archi vanno dalla transizione t alla transizione p.*

Nelle definizioni di posti e transizioni si suppone che l'insieme $P \cap T = \emptyset$ e l'insieme $P \cup T \neq \emptyset$.

In Fig. 3.1 è riportata una rete di Petri descritta delle seguenti equazioni:

$$P = \{P0, P1, P2, P3, P4\}$$

$$T = \{T0, T1, T2, T3\}$$

$$\boldsymbol{Pre} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{Post} = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

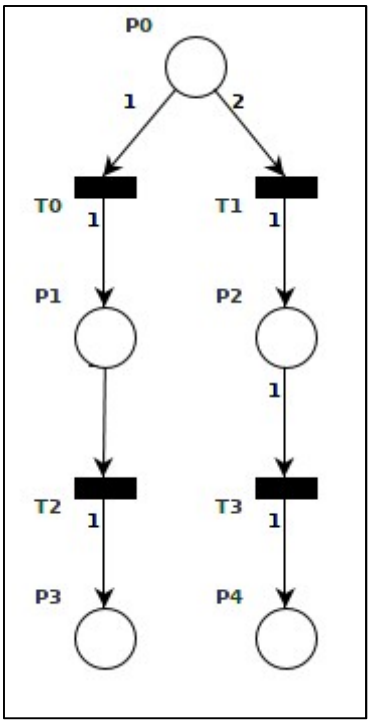

Figura 3-1 - Rete di Petri

La rete di Petri è un modello utilizzato per descrivere un sistema ad eventi discreti che evolve nel tempo, quindi è necessario definire una notazione per indicare lo stato del sistema in un dato instante. Per risolvere questo problema si introduce la *marcatura*.

Definizione 3.2: *Si definisce marcatura una funzione* $M: P \rightarrow \mathbb{N}$ *che assegna ad ogni posto un numero intero non negativo di marche (o gettoni).*

La marcatura individua lo stato in cui si trova una rete di Petri e viene indicata con dei token (gettoni) neri posti all'interno dei posti.

Definizione 3.3: *Data una rete di Petri* $\mathbb{N}$ *ed una marcatura iniziale* M_0 *(lo stato iniziale), si definisce rete marcata la quintupla* $\langle N, M_0 \rangle$.

In Fig. 3.2 viene riportata la rete di Petri della Fig.3.1 con la seguente marcatura.

$$\boldsymbol{m}_0 = [1 \quad 0 \quad 0 \quad 0 \quad 0]^T$$

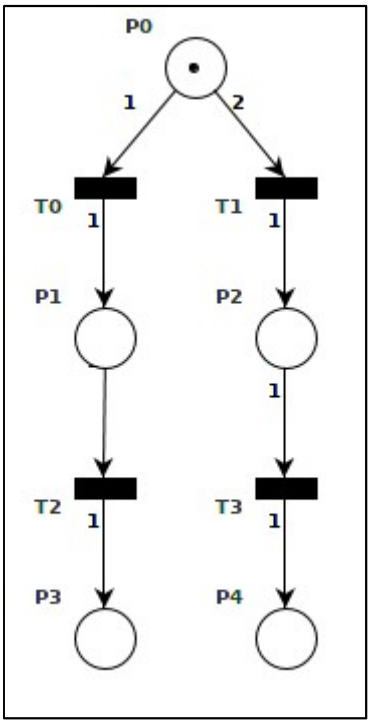

Figura 3-2 - Rete di Petri marcata

Definita la struttura e lo stato della rete di Petri, ci si pone il problema di analizzare l'evoluzione della rete nel tempo. Per prima cosa si introduce la *matrice d'incidenza*.

Definizione 3.4*: Si definisce matrice d'incidenza* $C: PxT \rightarrow \mathbb{N}$ *la matrice m x n definita come:*

$$C = Post - Pre \quad [3.2]$$

A differenza delle matrici di pre-incidenza e post-incidenza, la matrice C è una matrice che può contenere numeri interi negativi.

Gli interi negativi sono associati agli archi che partono dai posti e arrivano alle transizioni, gli interi positivi indicano gli archi che partono dalle transizioni e arrivano ai posti.

La matrice d'incidenza della rete in Fig. 3.2 è la seguente

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

Definita la rete marcata e la matrice d'incidenza, adesso è possibile spiegare come evolve una rete di Petri.

La rete di Petri è un modello per sistemi ed eventi discreti, quindi l'evoluzione di una rete è dettata dall'occorrenza di eventi (sincroni o asincroni) associati alle diverse transizioni.

Per far si che la transizione $t - iesima$ scatti è necessario che questa sia abilitata.

Definizione 3.5: *Una transizione t_j è abilitata dalla marcatura $\boldsymbol{M}$ se*

$$\boldsymbol{M} \geq \boldsymbol{Pre}(\cdot, t) \text{ [3.3]}$$

cioè se ogni posto $p \in P$ della rete che è collegata alla transizione t con un arco da p a t contiene un numero di marche pari o superiore a $\boldsymbol{Pre}(p, t)$.

In Fig. 3.2 data la marcatura iniziale M_0, la transizione t_0 è abilitata mentre la transizione t_1 non è abilitata dato che l'arco richiede un numero di marcatura pari a 2.

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

$$M' = M - Pre(\cdot, t) + Post(\cdot, t) = M + C(\cdot, t) \text{ [3.4]}$$

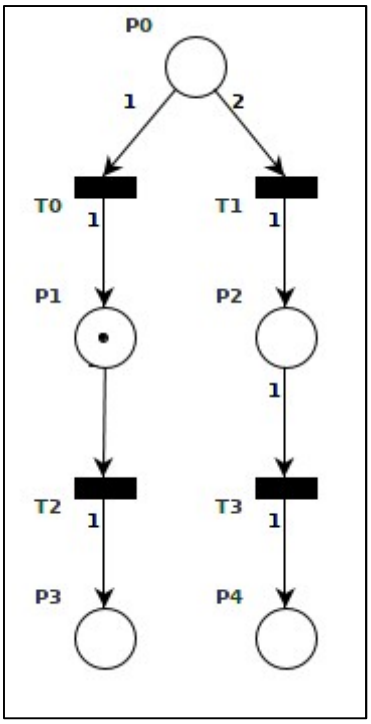

Figura 3-3 Rete di Petri marcata della figura 3-2 dopo lo scatto della transizione t_0

Una sequenza di transizioni $\sigma = t_{j_1}, t_{j_2}, \ldots, t_{j_r} \in T^*$ è abilitata da una marcatura M se: la transizione t_{j_1} è abilitata da M e il suo scatto porta a $M_1 = M + C\left(\cdot, t_{j_1}\right)$; la transizione t_{j_2} è abilitata da M_1 e il suo scatto porta a $M_2 = M_1 + C\left(\cdot, t_{j_2}\right)$; ecc.

Una sequenza abilitata σ viene anche detta sequenza di scatto e a essa corrisponde la traiettoria: $M[t_{j_1}\rangle M_1[t_{j_2}\rangle M_2 \ldots [t_{j_r}\rangle M_r$

Per indicare che la sequenza σ è abilitata da M e determina la marcatura M' si usa la notazione $M[\sigma\rangle M'$.

Data una rete $\langle N, M_0\rangle$ con insieme di transizioni $T = \{t_1, t_2, \ldots, t_n\}$e una sequenza di transizioni $\sigma \in T^*$, si definisce vettore caratteristico di σ (o anche vettore di scatto

associato a σ) il vettore $\boldsymbol{\sigma} \in \mathbb{N}^n$, la cui generica componente $\sigma - iesima = |\boldsymbol{\sigma}|_{t_i}$ indica quante volte la transizione t_i compare in σ.

Una conseguenza del vettore di scatto è la seguente:

Definizione 3.6: *Sia $\langle N, M_0 \rangle$ una rete marcata e sia C la sua matrice di incidenza. Se M è raggiungibile da M_0 attraverso il vettore di scatto σ vale:*

$$M = M_0 + C \cdot \boldsymbol{\sigma} \text{ [3.5]}$$

Definizione 3.7: *Una marcatura M è raggiungibile in $\langle N, M_0 \rangle$ se esiste una sequenza di scatto a tale che $M_0[\sigma\rangle M$. L'insieme di raggiungibilità di una rete marcata $\langle N, M_0 \rangle$ è l'insieme delle marcature che possono venir raggiunte a partire dalla marcatura iniziale, cioè l'insieme*

$$R(N, M_0) = \{M \in \mathbb{N}^n \mid \exists\, \sigma \in L(N, M_0) : M_0[\sigma\rangle M \} \text{ [3.6]}$$

Infine, per lo scopo di questa tesi viene riportata la definizione di rete limitata.

Definizione 3.8: *Un posto p è k-limitato in $\langle N, M_0 \rangle$ se per ogni marcatura raggiungibile $M \in R(N, M_0)$ vale $M(p) \leq k$*

Definizione 3.9: *Una rete marcata $\langle N, M_0 \rangle$ è k-limitata se ogni suo posto è k-limitato.*

3.2 Proprietà di non interferenza

Data una rete di Petri definita dall'equazione 3.1, è possibile suddividere l'insieme delle transizioni T nei due sottoinsiemi L e H, dove

- L indica l'insieme delle transizioni di basso livello, ossia tutte le transizioni che possono essere attivate anche da utenti esterni (ad esempio un intruso).
- H indica l'insieme delle transizioni di alto livello, ossia tutte le transizioni che possono essere attivate solo dall'utente principale.

Definiti gli insiemi L e H, è possibile individuare due sotto reti della rete principale:

- la *rete di alto livello*, ossia l'insieme dei posti e delle transizioni visibili dell'utente di alto livello;
- la *rete di basso livello*, ossia l'insieme dei posti e delle transizioni visibili dall'utente di basso livello.

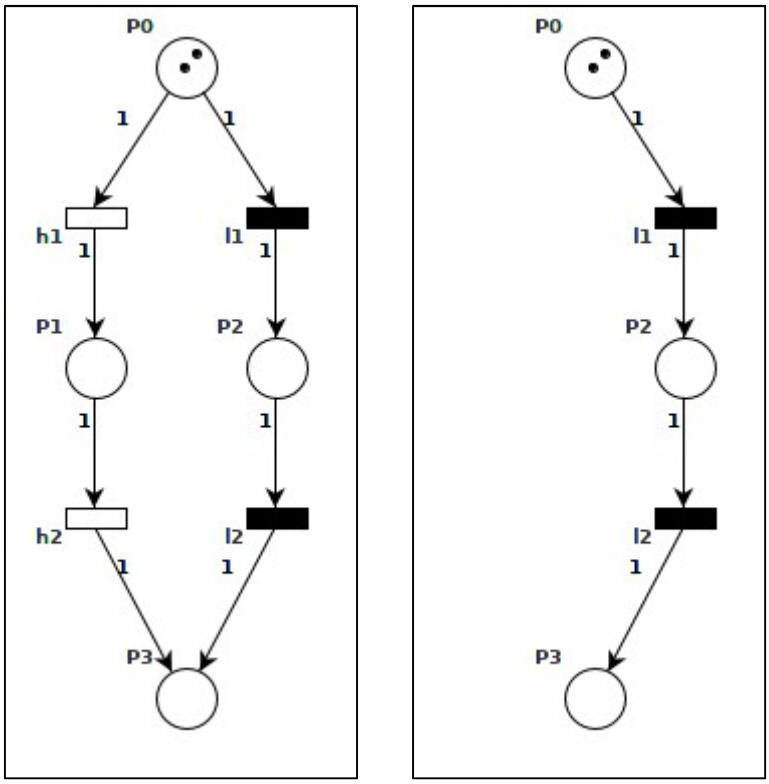

Figura 3-4 - Rete di Petri e sua sottorete di basso livello

Definita la rete di basso livello, ci si pone il problema di capire se un utente di basso livello possa carpire delle informazioni sulla struttura della rete ed interferire con il suo funzionamento avendo accesso solamente alle transizioni di basso livello. In pratica ci si chiede se la rete sia *non interferente*.

La definizione di non interferenza non è univoca, ma racchiude un gruppo di proprietà con condizioni sempre più stringenti.

In questa tesi ci si focalizzerà sulla proprietà di *Strong Non-deterministic Non-Interference* (SNNI).

Una rete di Petri si definisce SNNI se, scattate tutte le possibili transizioni di basso livello abilitate a partire dalla marcatura iniziale $\boldsymbol{M_0}$, nessuna transizione di alto livello può abilitare un ulteriore transizione di basso livello[1].

La rete in figura 3.4 è un esempio di rete di Petri che verifica la proprietà di SNNI in quanto né h_1 o h_2 potranno abilitare ulteriori scatti delle transizioni l_1 o l_2 oltre le due consentite.

Invece la rete in Fig. 3.5 non è SNNI perché data la marcatura iniziale M_0 , supponendo che la transizione h_2 non scatti mai, la transizione l_1 può scattare al massimo tre volte

[1] La definizione formale della proprietà SNNI fa uso della teoria dei linguaggi che esula dello scopo di questa tesi.

volta. Se scattasse una quarta volta, l'utente di basso livello capirebbe che una o più transizioni[2] di alto livello sono scattate, avendo una perdita di informazione.

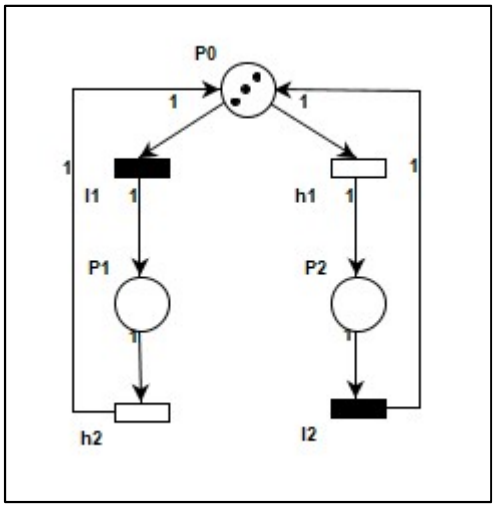

Figura 3-5 Rete di Petri non SNNI

Mentre per una rete non limitata la verifica della proprietà SNNI è un problema non decidibile, per le reti limitate vale il seguente lemma dimostrato in [2]:

Teorema 3.1: *Data una rete limitata* $S = \langle N, \boldsymbol{m}_0 \rangle$*, sia* $\hat{\sigma}_t$ *uguale a*

$$\hat{\sigma}_t = max \sum_{i=1}^{J} \boldsymbol{\sigma}_i(t) \quad [3.7]$$

Sottoposto a

$$\begin{cases} \boldsymbol{m}_0 \geq \boldsymbol{Pre}_L \cdot \boldsymbol{\sigma}_1 \\ \boldsymbol{m}_0 + \boldsymbol{C}_L \cdot \boldsymbol{\sigma}_1 \geq \boldsymbol{Pre}_L \cdot \boldsymbol{\sigma}_2 \\ \dots \\ \boldsymbol{m}_0 + \boldsymbol{C}_L \cdot \sum_{i=1}^{J-1} \boldsymbol{\sigma}_i \geq \boldsymbol{Pre}_L \cdot \boldsymbol{\sigma}_J \\ \boldsymbol{m}_0 + \boldsymbol{C}_L \cdot \sum_{i=1}^{J} \boldsymbol{\sigma}_i \geq 0 \\ \boldsymbol{\sigma}_i \in \mathbb{N}^{n_L}, \quad i = 1,2,\dots,J \end{cases} \quad [3.8]$$

Con $t \in L$ *e* $J \geq J_{min}$*. Si considerino i due seguenti problemi PLI*

$$min \sum_{i=1}^{J} \sum_{t_h \in H} x_i(t_h) \quad [3.9]$$

[2] L'utente di basso livello, non conoscendo la struttura della rete di alto livello, non può escludere che siano state attivate più transizioni di alto livello.

Soggetto a

$$\boldsymbol{\chi}\,(\boldsymbol{m}_0, \widehat{\sigma_t}) : \begin{cases} \boldsymbol{m}_0 \geq \boldsymbol{Pre} \cdot \boldsymbol{x}_1 \\ \boldsymbol{m}_0 + \boldsymbol{C} \cdot \boldsymbol{x}_1 \geq \boldsymbol{Pre} \cdot \boldsymbol{x}_2 \\ \dots \\ \boldsymbol{m}_0 + \boldsymbol{C} \cdot \sum_{i=1}^{J-1} \boldsymbol{x}_i \geq \boldsymbol{Pre} \cdot \boldsymbol{x}_J \\ \boldsymbol{m}_0 + \boldsymbol{C} \cdot \sum_{i=1}^{J} \boldsymbol{x}_i \geq 0 \\ \sum_{i=1}^{J} \boldsymbol{x}_i \geq \widehat{\sigma_t} + 1 \\ \boldsymbol{x}_i \in \mathbb{N}^{n_L + n_H}, \qquad i = 1,2 \dots, J \end{cases} \quad [3.10]$$

e

$$min \sum_{i=1}^{J} \sum_{t_l \in L} y_i(t_l) \quad [3.11]$$

Soggetto a

$$\boldsymbol{\Upsilon}\,(\boldsymbol{m}_0, \widehat{\sigma_t}) : \begin{cases} \boldsymbol{m}_0 \geq \boldsymbol{Pre} \cdot \boldsymbol{y}_1 \\ \boldsymbol{m}_0 + \boldsymbol{C} \cdot \boldsymbol{y}_1 \geq \boldsymbol{Pre} \cdot \boldsymbol{y}_2 \\ \dots \\ \boldsymbol{m}_0 + \boldsymbol{C} \cdot \sum_{i=1}^{J-1} \boldsymbol{y}_i \geq \boldsymbol{Pre} \cdot \boldsymbol{y}_J \\ \boldsymbol{m}_0 + \boldsymbol{C} \cdot \sum_{i=1}^{J} \boldsymbol{y}_i \geq 0 \\ \sum_{i=1}^{J} \boldsymbol{y}_i \geq \widehat{\sigma_t} \\ \boldsymbol{y}_i \in \mathbb{N}^{n_L + n_H}, \qquad i = 1,2 \dots, J \end{cases} \quad [3.12]$$

Il sistema S è SNNI se e solo se le due seguenti condizioni valgono per tutte le $t \in L$*:*

1. *il problema PLI* [3.9] - [3.10] *non ammette soluzione o, se risolvibile, la funzione obiettivo* $min \sum_{i=1}^{J} \sum_{t_h \in H} x_i(t_h) = 0$
2. *la soluzione del problema* [3.11] - [3.12] $\widehat{\boldsymbol{y}}_1, \dots, \widehat{\boldsymbol{y}}_J \in \mathbb{N}^{n_L + n_H}$ *è tale che* $min \sum_{i=1}^{J} \sum_{t_l \in L} y_i(t_l) = 0$

Il lemma è costituito da tre problemi di ottimizzazione.

Il problema di ottimizzazione [3.7] - [3.8] ricerca, per ogni transizione di basso livello, il massimo vettore caratteristico che può essere abilitato dallo stato iniziale M_0.

Il problema di ottimizzazione [3.9] - [3.10] verifica che nessuna transizione di alto livello possa attivare un ulteriore scatto di una transizione di basso livello dopo aver attivato la sequenza di scatto calcolata nel primo problema di ottimizzazione.

Il problema di ottimizzazione [3.11] - [3.12] verifica che, per nessuna transizione di basso livello, questa possa essere sostituita da una transizione di alto livello o un'altra transizione di basso livello.

4 Specifiche del problema e risoluzione

Questo capitolo costituisce il corpo della tesi.

Inizialmente saranno definite le specifiche e i requisiti del progetto.

Successivamente saranno descritti tutti gli strumenti utilizzati nella tesi.

Infine, verrà spiegato il meccanismo di ogni funzione creata per questa tesi.

4.1 Pseudo codice per la sintesi di una rete SNNI

In [2] è stato proposto il seguente algoritmo che, data una rete di Petri limitata che non verifichi la proprietà di SNNI ed un insieme PS (l'insieme delle transizioni di alto livello che possono essere disabilitate), è in grado di verificare se il problema ammette una soluzione. In caso affermativo la funzione restituisce l'insieme più piccolo delle transizioni di PS da disabilitare per ottenere una rete di Petri SNNI.

Input $S = \langle N, \boldsymbol{m}_0 \rangle$, $PS \subseteq H$, T, and $\hat{\sigma}_t$ for all $\bar{t} \in T$

1. $NS = (L \cup H) \setminus PS$
2. **for all** $\bar{t}_l \in T$ **do**
   - 2.1. **solve** the ILP problem $\min \sum_{i=1}^{J} \sum_{t \in L \cup PS} \boldsymbol{y}_i(t)$ subject to set of constrains $Y(\boldsymbol{m}_0, \hat{\sigma}_{\bar{t}})$
   - 2.2. **let** $\widehat{\boldsymbol{y}}_i$, with $i = 1, \dots, J$, the solution of the problem solved at step 2.1
   - 2.3. **if** $\sum_{i=1}^{J} \sum_{t \in NS \cap H} \widehat{\boldsymbol{y}}_j(t) > 0$ **then** Problem 1 does not admit a solution and the algorithm **terminates**
   - 2.4. **solve** the ILP problem $\min \sum_{i=1}^{J} \sum_{t \in PS} \boldsymbol{x}_i(t)$ subject to set of constrains $X(\boldsymbol{m}_0, \hat{\sigma}_{\bar{t}})$
   - 2.5. **let** $\widehat{\boldsymbol{x}}_i$, with $i = 1, \dots, J$, the solution of the problem solved at step 2.4
   - 2.6. **if** $\sum_{i=1}^{J} \sum_{t \in NS \cap H} \widehat{\boldsymbol{x}}_j(t) > 0$ **then** Problem 1 does not admit a solution and the algorithm **terminates**

   end for
3. **for all** $\bar{t}_l \in T$ **do**
   - 3.1. $D_{\bar{t}_l} = \emptyset$
   - 3.2. $CS = PS$
   - 3.3. **solve** the ILP problem $\min \sum_{i=1}^{J} \sum_{t \in L \cup D_{\bar{t}_l}} \boldsymbol{y}_i(t)$ subject to set of constrains $Y(\boldsymbol{m}_0, \hat{\sigma}_{\bar{t}})$
   - 3.4. **let** $\widehat{\boldsymbol{y}}_i$, with $i = 1, \dots, J$, the solution of the problem solved at step 3.3
   - 3.5. **if** $\sum_{i=1}^{J} \sum_{t \in CS} \widehat{\boldsymbol{y}}_j(t) > 0$ **then**
     - 3.5.1. $D_{\bar{t}_l} = D_{\bar{t}_l} \cup \{t \in CS | \widehat{\boldsymbol{y}}_i(t) > 0\}$
     - 3.5.2. $CS = CS \setminus \{t \in CS | \widehat{\boldsymbol{y}}_i(t) > 0\}$
     - 3.5.3. **if** CS is not empty **then go to** Step 3.3 **else** Problem 1 admits the solution $D = PS$ and the algorithm **terminates**
   - 3.6. **solve** the ILP problem $\min \sum_{i=1}^{J} \sum_{t \in D_{\bar{t}_l}} \boldsymbol{x}_i(t)$ subject to the constrains $X(\boldsymbol{m}_0, \hat{\sigma}_{\bar{t}})$
   - 3.7. **let** $\widehat{\boldsymbol{x}}_i$, with $i = 1, \dots, J$, the solution of the problem solved at step 3.6
   - 3.8. **if** $\sum_{i=1}^{J} \sum_{t \in CS} \widehat{\boldsymbol{x}}_j(t) > 0$ **then**
     - 3.8.1. $D_{\bar{t}_l} = D_{\bar{t}_l} \cup \{t \in CS | \widehat{\boldsymbol{x}}_i(t) > 0\}$
     - 3.8.2. $CS = CS \setminus \{t \in CS | \widehat{\boldsymbol{x}}_i(t) > 0\}$
     - 3.8.3. **if** CS is not empty **then go to** Step 3.6 **else** Problem 1 admits the solution $D = PS$ and the algorithm **terminates**

   end for
4. The solution to the Problem 1 is given by $D = \bigcup_{\bar{t}_l \in T} D_{\bar{t}_l}$

L'obiettivo della tesi è di implementare lo pseudo-codice in una funzione MATLAB.

4.2 Strumenti utilizzati

Per l'implementazione della funzione principale e le funzioni di supporto è stato utilizzato il linguaggio di programmazione MATLAB.

MATLAB (abbreviazione di Matrix Laboratory) è un ambiente per il calcolo numerico e l'analisi statistica scritto in C, che comprende anche l'omonimo linguaggio di programmazione creato dalla MathWorks. [3]

Per modellare e risolvere i problemi di ottimizzazione è stato utilizzato il toolbox YALMIP.

YALMP è un parser che creare un layer di astrazione tra il modello del problema di ottimizzazione e la sua implementazione consentendo all'utente di concentrarsi sul problema di ottimizzazione in maniera astratta.

Per risolvere i problemi di ottimizzazione è necessario indicare il solver da utilizzare. In questa tesi è stato utilizzato il toolkit GLPK.

GLPK (GNU Linear Programming Kit) è una libreria software scritta in ANSI C ed è utilizzabile per risolvere problemi di programmazione lineare sia continui (LP) che misto interi (MILP). [4]

Infine, per l'utilizzo delle funzioni di GLPK è necessario utilizzare un'interfaccia tra MATLAB e GLPK. In questa tesi è stato utilizzato GLPKMEX.

4.3 Implementazione del codice in MATLAB

Prima di spiegare il funzionamento del codice è bene definire la struttura della funzione.

Il codice è diviso in tre sezioni:

- **Entry-point**: *makeSNNI.m*
- **Funzioni principali**: *isSNNI.m*, *getSNNI.m*
- **Funzioni di supporto:** *getConstrains.m*, *getSigmaConstrains.m*, *solveXProblem.m*, *solveYProblem.m*, *test_SNNI.m*.

Inizialmente verrà descritta la funzione main, successivamente verranno descritte le funzioni principali, ed infine le funzioni di supporto.

4.3.1 makeSNNI.m

La funzione makeSNNI.m costituisce il main del programma. Di seguito è riportato il codice implementativo.

```
1.  function [ D ] = makeSNNI(Pre, Post, m0, L, J, PS)
2.  addpath(genpath('utility'));
3.
4.  C = Post - Pre;
5.
6.  [T,sigma_t] = isSNNI(Pre, C, m0, L, J);
7.
8.  if(isempty(T))
9.      D = [];
10. else
11.     D = getSNNI(Pre, C, m0, L, J, PS, T, sigma_t);
12. end
13. end
```

La funzione richiede i seguenti dati di input:

- $\boldsymbol{Pre}$, ossia la matrice di pre-incidenza;
- $\boldsymbol{Post}$, ossia la matrice di post-incidenza;
- $\boldsymbol{m}_0$, ossia la marcatura iniziale della rete di Petri;
- L, ossia l'insieme delle transizioni di basso livello;
- J, ossia il numero minimo di transizioni per raggiungere qualsiasi marcatura dell'insieme di raggiungibilità;
- PS, ossia l'insieme delle transizioni di alto che possono essere disabilitate.

La funzione restituisce l'insieme minimo D, delle transizioni da disabilitare per rendere la rete SNNI.

Alla riga 2 la funzione aggiunge alla path di MATLAB tutte le path delle funzioni di supporto in modo da poter essere richiamate dalle funzioni principali.

Alla riga 4 viene calcolata la matrice d'incidenza della rete di Petri.

Alla riga 6 viene richiamata la funzione *isSNNI.m*, la quale verifica che la rete di Petri sia SNNI restituendo le soluzioni $\hat{\sigma}_t$ del problema PLI [3.7] - [3.8] e l'insieme delle transizioni di basso livello T per cui non sono verificate tale problema PLI.

Infine, dalla riga 8 alla riga 12 è presente uno if-else che valuta l'insieme T:

- nel caso in cui $T = \emptyset$, la rete sarà SNNI quindi l'insieme delle transizioni da disabilitare $D = \emptyset$
- nel caso in cui $T \neq \emptyset$, viene richiamata la funzione *getSNNI.m*, la quale calcola l'insieme minimo di transizioni di alto livello da disabilitare per ottenere una rete SNNI

4.3.2 isSNNI.m

La funzione isSNNI.m è adibita al controllo della proprietà di SNNI. Precisamente verifica, per ogni transizione di basso livello, se è verificata la proprietà di SNNI, restituendo l'insieme T, ossia l'insieme delle transizioni per cui non sono risolvibili i problemi [3.9] – [3.10] e [3.11] – [3.12].

Di seguito è riportata l'implementazione.

```
function [T, sigma_t] = isSNNI(Pre, C, m0, L, J)
settings = sdpsettings('solver', 'glpk');

T = [];

PreL = Pre(:,L);
CL = C(:,L);

[~,num_L_transitions] = size(CL);
[~,num_transitions] = size(C);

H = setxor(1:num_transitions, L);

sigma_t = [];

for i = 1:num_L_transitions
    sigma = intvar(num_L_transitions,J);

    temp = sum(sigma,2);
    temp = temp(i);

    Constrains = getSigmaConstrains(PreL, CL, m0, J, sigma);

    optimize(Constrains, -temp, settings);

    sigma_t = [sigma_t,value(temp)];
end

for i = L
     [x,solution_x] =  solveXProblem(Pre, C, m0, J, sigma_t, H, i);
     [y,~] =  solveYProblem(Pre, C, m0, J, sigma_t, L, i);

    if((solution_x.problem == 0 && sum(sum(x(H,:)),2) ~= 0) ||
sum(sum(y(H,:)),2) ~= 0)
        T = [T,i];
    end
end
end
```

La funzione richiede i seguenti dati di input:

- $\boldsymbol{Pre}$, ossia la matrice di pre-incidenza;
- $\boldsymbol{C}$, ossia la matrice di incidenza;
- $\boldsymbol{m}_0$, ossia la marcatura iniziale della rete di Petri;
- L, ossia l'insieme delle transizioni di basso livello;
- J, ossia il numero minimo di transizioni per raggiungere qualsiasi marcatura dell'insieme di raggiungibilità.

La funzione restituisce i seguenti dati di output:

- T, ossia l'insieme
- sigma_t, ossia le soluzioni del problema PLI [3.7] - [3.8].

Alla riga 2 viene settato il solver che YALMIP deve utilizzare (in questo caso GLPK).

Alla riga 4-14 vengono inizializzate le variabili utilizzate nella funzione.

Alle righe 16-27, per ogni transizione di basso livello, viene risolto il problema PLI [3-7] – [3.8] e la soluzione viene aggiunta al vettore sigma_t.

Infine, alle righe 29-36 per ogni transizione di basso livello vengono svolti i problemi PLI [3.9] – [3.10] e [3.11] – [3.12] e viene verificato che le soluzioni verifichino le condizioni del **Teorema 1**. In caso negativo, la transizione $t - iesima$ viene aggiunta al vettore T.

4.3.3 getSNNI.m

La funzione getSNNI.m è l'implementazione dell'algoritmo presentato in [2]. Data ingresso una rete non SNNI ed un insieme di transizioni di alto livello che possono essere disabilitate, restituisce l'insieme minimo delle transizioni da disabilitare.

Di seguito è riportata l'implementazione.

```
function [ D ] = getSNNI(Pre, C, m0, L, J, PS, T, sigma_t)

[~, transitions_num] = size(C);

H = setxor(1:transitions_num, L);
NS = setxor(union(L,H),PS);
D = [];


for i = T
    [y,~] =  solveYProblem(Pre, C, m0, J, sigma_t, union(L,PS), i);
    if( sum( sum( y(intersect(NS,H),1:J) ) ,2) > 0 )
        error('Problem does not admin solution')
    End

    [x,~] = solveXProblem(Pre, C, m0, J, sigma_t, PS, i);
    if( sum( sum( x(intersect(NS,H),1:J) ) ,2) > 0 )
        error('Problem does not admin solution')
    end
end

for i = T
    Dt = [];
    CS = PS;
    [y,~] =  solveYProblem(Pre, C, m0, J, sigma_t, union(L,Dt), i);
    if( sum( sum( y(CS,:) ) ) > 0 )
        for j = CS
            if ( sum( y(j,:) ) > 0 )
                Dt = union(Dt,j);
                CS = setxor(CS,j);
            end
            if ( isempty(CS) )
                D = PS;
                return;
            end
        end
    end
    [x,~] = solveXProblem(Pre, C, m0, J, sigma_t, Dt, i);
    if( sum( sum( x(CS,:) ) ) > 0 )
        for j = CS
            if ( sum ( x(j,:) ) > 0 )
                Dt = union(Dt,j);
                CS = setxor(CS,j);
            end
            if ( isempty(CS) )
                D = PS;
                return;
            end
        end
    end
    D = union(D,Dt);
```

La funzione richiede i seguenti dati di input:

- $\boldsymbol{Pre}$, ossia la matrice di pre-incidenza;
- $\boldsymbol{Post}$, ossia la matrice di post-incidenza;
- $\boldsymbol{m}_0$, ossia la marcatura iniziale della rete di Petri;
- L, ossia l'insieme delle transizioni di basso livello;
- J, ossia il numero minimo di transizioni per raggiungere qualsiasi marcatura dell'insieme di raggiungibilità;
- PS, ossia l'insieme delle transizioni di alto che possono essere disabilitate;
- T, ossia l'insieme delle transizioni di basso livello per cui non sono verificate le condizioni del Teorema 1;
- sigma_t, ossia il vettore delle soluzioni del problema PLI [3.7] – [3.8].

La funzione restituisce l'insieme delle transizioni di alto livello da disabilitare per ottenere la rete SNNI.

Dalla riga 3 alla riga 9 vengono inizializzate le variabili utilizza allo svolgimento della funzione.

Dalla riga 10 alla riga 20 vengono risolti nuovamente i problemi [3.9] – [3.10] e [3.11] – [3.12] verificando che questi problemi ammettano soluzione per la scelta di PS[3].

Dalla riga 22 alla riga 51 vengono calcolate le transizioni da disabilitare risolvendo iterativamente i problemi [3.9] – [3.10] e [3.11] – [3.12] aggiungendo all'insieme D, ad ogni iterazione, le transizioni di alto livello per cui non sono verificate condizioni del Teorema 1.

4.3.4 getSigmaConstrains.m

La funzione getSigmaConstrains.m definisce i vincoli del problema [3.7] – [3.8].

Di seguito è riportato il codice implementativo.

[3] Con un'errata scelta dell'insieme PS potrebbe non essere possibile rendere una rete di Petri SNNI. Nell'esempio 1 vi è una dimostrazione di tale situazione. Se risulta $PS = H$ allora è sempre possibile trovare una soluzione ai due problemi, nel peggiore dei casi bisogna disabilitare tutte le transizioni di alto livello.

```
1.  function Constrains = getSigmaConstrains(PreL, CL, m0, J, sigma)
2.
3.  [~, transitions_num] = size(CL);
4.
5.  Constrains = [];
6.
7.  for i = 1:transitions_num
8.      for j = 1:J
9.          Constrains = [Constrains, sigma(i,j) >= 0];
10.     end
11. end
12.
13. for i = 1:J+1
14.     if (i == 1)
15.         Constrains = [Constrains, m0 >= PreL * sigma(:,i)];
16.     elseif (i == J+1)
17.         Constrains = [Constrains, m0 + CL * sum(sigma(:,1:i-1),2)
    >= 0];
18.     else
19.         Constrains = [Constrains, m0 + CL * sum(sigma(:,1:i-1),2)
    >= PreL * sigma(:,i)];
20.     end
21. end
22.
23. end
```

La funzione richiede i seguenti dati di input:

- $\boldsymbol{Pre_L}$, ossia la matrice di pre-incidenza della rete di basso livello;
- $\boldsymbol{C_L}$, ossia la matrice di incidenza della rete di basso livello;
- $\boldsymbol{m}_0$, ossia la marcatura iniziale della rete di Petri;
- J, ossia il numero minimo di transizioni per raggiungere qualsiasi marcatura dell'insieme di raggiungibilità;
- sigma, ossia le variabili intere del problema PLI [3.7] – [3.8].

La funzione restituisce i vincoli del problema PLI [3.7] – [3.8].

Alla riga 3 viene calcolato il numero di transizioni di basso livello.

Alla riga 5 viene inizializzato il vettore che conterrà i vincoli.

Dalla riga 7 alla riga 11 vengono calcolati i vincoli $\boldsymbol{\sigma}_i \in \mathbb{N}^{n_L},\ i = 1,2,\dots,J$

Infine, dalla riga 13 alla riga 21 vengono calcolati i seguenti vincoli:

- Se i = 1 viene calcolata il vincolo $\boldsymbol{m}_0 \geq \boldsymbol{Pre}_L \cdot \boldsymbol{\sigma}_1$
- Se i = J + 1 viene calcolata il vincolo $\boldsymbol{m}_0 + \boldsymbol{C}_L \cdot \sum_{i=1}^{J} \boldsymbol{\sigma}_i \geq 0$
- Altrimenti viene calcolata il vincolo $\boldsymbol{m}_0 + \boldsymbol{C}_L \cdot \sum_{i=1}^{z-1} \boldsymbol{\sigma}_i \geq \boldsymbol{Pre}_L \cdot \boldsymbol{\sigma}_z$

4.3.5 getConstrains.m

La funzione getConstrains.m definisce i vincoli del problema [3.9] – [3.10] e del problema [3.11] - [3.12].

Di seguito è riportato il codice implementativo.

```
1.  function Constrains = getConstrains(Pre, C, m0, J, z)
2.
3.  [~,transitions_num] = size(C);
4.
5.  Constrains = [];
6.
7.  for i = 1:transitions_num
8.      for j = 1:J
9.          Constrains = [Constrains, z(i,j)];
10.     end
11. end
12.
13. for i = 1:J+1
14.     if (i == 1)
15.         Constrains = [Constrains, m0 >= Pre * z(:,i)];
16.     elseif (i == J+1)
17.         Constrains = [Constrains, m0 + C * sum(z(:,1:i-1),2) >= 0];
18.     else
19.         Constrains = [Constrains, m0 + C * sum(z(:,1:i-1),2) >= Pre
    * z(:,i)];
20.     end
21. end
22.
23. end
```

La funzione richiede i seguenti dati di input:

- $\boldsymbol{Pre}$, ossia la matrice di pre-incidenza della rete di Petri;
- $\boldsymbol{C}$, ossia la matrice di incidenza della rete di Petri;
- $\boldsymbol{m}_0$, ossia la marcatura iniziale della rete di Petri;
- J, ossia il numero minimo di transizioni per raggiungere qualsiasi marcatura dell'insieme di raggiungibilità;
- z, ossia le variabili intere del problema PLI [3.9] – [3.10] o [3.11] - [3.12].

La funzione restituisce i vincoli del problema PLI [3.9] – [3.10] o [3.11] - [3.12].

Alla riga 3 viene calcolato il numero di transizioni della rete di Petri.

Alla riga 5 viene inizializzato il vettore che conterrà i vincoli.

Dalla riga 7 alla riga 11 vengono calcolati i vincoli $\boldsymbol{z}_i \in \mathbb{N}^{n_L+n_H},\ i = 1,2,\dots,J$

Infine, dalla riga 13 alla riga 21 vengono calcolati i seguenti vincoli:

- Se i = 1 viene calcolata il vincolo $\boldsymbol{m}_0 \geq \boldsymbol{Pre} \cdot \boldsymbol{z}_1$

- Se i = J + 1 viene calcolata il vincolo $\boldsymbol{m}_0 + \boldsymbol{C} \cdot \sum_{i=1}^{J} \boldsymbol{z}_i \geq 0$
- Altrimenti viene calcolata il vincolo $\boldsymbol{m}_0 + \boldsymbol{C} \cdot \sum_{i=1}^{p-1} \boldsymbol{z}_i \geq \boldsymbol{Pre} \cdot \boldsymbol{z}_p$

A differenza dei vincoli del problema PLI [3.7] – [3.8], i problemi [3.9] – [3.10] e [3.11] – [3.12] hanno un ulteriore vincolo che verrà aggiunto nelle funzioni solveXProblem.m e solveYProblem.m.

4.3.6 solveXProblem.m

La funzione solveXProblem.m si occupa di calcolare e risolvere il problema PLI [3.9] – [3.10].

Di seguito è riportato il codice implementativo.

```
1. function [x,solution] = solveXProblem (Pre, C, m0, J, sigma_t,
   sum_set, t)
2.
3. settings = sdpsettings('solver', 'glpk');
4.
5. [~, transitions_num] = size(C);
6.
7. x = intvar(transitions_num,J);
8.
9. f_x = sum(sum(x(sum_set,:)));
10.
11.Constrains = getConstrains(Pre, C, m0, J, x);
12.Constrains = [Constrains, sum(x(t,:),2) >= sigma_t(t) + 1];
13.
14.solution = optimize(Constrains, f_x, settings);
15.
16.x = value(x);
17.
18. end
```

La funzione richiede i seguenti dati di input:

- $\boldsymbol{Pre}$, ossia la matrice di pre-incidenza della rete di Petri;
- $\boldsymbol{C}$, ossia la matrice di incidenza della rete di Petri;
- $\boldsymbol{m}_0$, ossia la marcatura iniziale della rete di Petri;
- J, ossia il numero minimo di transizioni per raggiungere qualsiasi marcatura dell'insieme di raggiungibilità;
- **sigma_t,** ossia le soluzioni del problema PLI [3.7] – [3.8];
- **sum_set**, ossia l'insieme delle transizioni da sommare nella funzione obiettivo;
- **t**, ossia la transizione $i - esima$ per calcolare il vincolo aggiuntiva[4].

[4] Il vincolo aggiuntivo è $\sum_{i=1}^{J} \boldsymbol{x}_i \geq \hat{\sigma}_t(t) + 1$

La funzione restituisce:

- $\boldsymbol{x}$, ossia le soluzioni del PLI [3.9] – [3.10]
- **solution**, ossia i risultati della risoluzione del problema PLI.

Alla riga 3 viene settato il solver da utilizzare, ossia il solver GLPK.

Alla riga 5 viene calcolato il numero di transizioni della rete di Petri.

Alla riga 7 viene creato una matrice di interi con un numero di righe pari al numero di transizioni della rete e un numero di colonne pari a J.

Alla riga 9 viene definita la funzione obiettivo [3.9], con la differenza che l'insieme delle transizioni di su cui effettuare la somma viene definita attraverso **sum_set**. Questo perché, come si vedrà successivamente in getSNNI.m, consente di risolvere più problemi di ottimizzazione con la stessa funzione.

Alle righe 11 e 12 viene richiamata la funzione getConstrains.m descritta precedentemente e viene aggiunto il vincolo $\sum_{i=1}^{J} \boldsymbol{x}_i \geq \widehat{\sigma}_t(t) + 1$.

Alla riga 14 viene effettivamente risolto il problema PLI.

Infine, alla riga 16 viene convertito vettore $\boldsymbol{x}$ in un vettore di interi[5].

4.3.7 solveYProblem.m

La funzione solveYProblem.m si occupa di calcolare e risolvere il problema PLI [3.11] – [3.12].

Di seguito è riportato il codice implementativo.

[5] La funzione intvar istanzia una classe di oggetti che vengono utilizzati dalla funzione optimize. La funzione value si occupa di estrarre dalla matrice $\boldsymbol{x}$ effettivamente la matrice di interi.

```
1.  function [y,solution] = solveYProblem (Pre, C, m0, J, sigma_t,
    sum_set, t)
2.
3.  settings = sdpsettings('solver', 'glpk');
4.
5.  [~, transitions_num] = size(C);
6.
7.  y = intvar(transitions_num,J);
8.
9.  f_y = sum(sum(y(sum_set,:)));
10.
11. Constrains = getConstrains(Pre, C, m0, J, y);
12. Constrains = [Constrains, sum(y(t,:),2) >= sigma_t(t)];
13.
14. solution = optimize(Constrains, f_y, settings);
15.
16. y = value(y);
17.
18. end
```

La funzione richiede i seguenti dati di input:

- $\boldsymbol{Pre}$, ossia la matrice di pre-incidenza della rete di Petri;
- $\boldsymbol{C}$, ossia la matrice di incidenza della rete di Petri;
- $\boldsymbol{m}_0$, ossia la marcatura iniziale della rete di Petri;
- J, ossia il numero minimo di transizioni per raggiungere qualsiasi marcatura dell'insieme di raggiungibilità;
- **sigma_t,** ossia le soluzioni del problema PLI [3.7] – [3.8];
- **sum_set**, ossia l'insieme delle transizioni da sommare nella funzione obiettivo;
- **t**, ossia la transizione $i-esima$ per calcolare il vincolo aggiuntiva[6].

La funzione restituisce:

- $\boldsymbol{y}$, ossia le soluzioni del PLI [3.11] – [3.12]
- **solution**, ossia i risultati della risoluzione del problema PLI.

Alla riga 3 viene settato il solver da utilizzare, ossia il solver GLPK.

Alla riga 5 viene calcolato il numero di transizioni della rete di Petri.

Alla riga 7 viene creato una matrice di interi con un numero di righe pari al numero di transizioni della rete e un numero di colonne pari a J.

Alla riga 9 viene definita la funzione obiettivo [3.9], con la differenza che l'insieme delle transizioni di su cui effettuare la somma viene definita attraverso **sum_set**. Questo perché,

[6] Il vincolo aggiuntivo è $\sum_{i=1}^{J} \boldsymbol{y}_i \geq \hat{\sigma}_t(t)$

come si vedrà successivamente in getSNNI.m, consente di risolvere più problemi di ottimizzazione con la stessa funzione.

Alle righe 11 e 12 viene richiamata la funzione getConstrains.m descritta precedentemente e viene aggiunto il vincolo $\sum_{i=1}^{J} \boldsymbol{y}_i \geq \widehat{\sigma_t}(t)$.

Alla riga 14 viene effettivamente risolto il problema PLI.

Infine, alla riga 16 viene convertito vettore $\boldsymbol{y}$ in un vettore di interi.

4.3.8 test_SNNI.m

La funzione test_SNNI.m verifica che la rete in ingresso sia SNNI.

Di seguito è riportato il codice implementativo

```
function [] = test_SNNI(Pre, Post, m0, L, J)
addpath(genpath('utility'));
settings = sdpsettings('solver', 'glpk');

C = Post - Pre;
T = [];
PreL = Pre(:,L);
CL = C(:,L);
[~,num_L_transitions] = size(CL);
[~,num_transitions] = size(C);
H = setxor(1:num_transitions, L);

sigma_t = [];

for i = 1:num_L_transitions
    sigma = intvar(num_L_transitions,J);
    temp = sum(sigma,2);
    temp = temp(i);
    Constrains = getSigmaConstrains(PreL, CL, m0, J, sigma);
    optimize(Constrains, -temp, settings);
    sigma_t = [sigma_t,value(temp)];
end

for i = L
    [x,solution_x] =  solveXProblem(Pre, C, m0, J, sigma_t, H, i);
    [y,~] =  solveYProblem(Pre, C, m0, J, sigma_t, L, i);
    if((solution_x.problem == 0 && sum(sum(x(H,:)),2) ~= 0) ||
sum(sum(y(H,:)),2) ~= 0)
        T = [T,i];
    end
end

if(isempty(T))
    display('Petri Net is SNNI.');
else
    display('Petri Net is not SNNI.');
end
end
```

La struttura della funzione è analoga alla funzione isSNNI.m, differisce da quest'ultima solo nei valor restituiti[7].

Calcolato il vettore *T*, la funzione verifica se il vettore è vuoto:

- in caso affermativo le condizioni del Teorema 1 sono verificate, e quindi la rete è SNNI.

[7] La funzione isSNNI.m restituisce l'insieme *T* e il vettore delle soluzioni del problema [3.7] – [3.8]

- in caso negativo le condizioni del Teorema 1 non sono verificate, e quindi la rete non è SNNI.

5 Esempi

In questo capitolo saranno presentati due casi d'uso della funzione implementata evidenziando come un'errata scelta dell'insieme PS renda impossibile ottenere una rete SNNI.

Inizialmente la rete verrà analizzata dalla funzione test_SNNI.m per verificare che la rete non rispetti la proprietà di SNNI.

Successivamente, la rete sarà posta in ingresso alla funzione makeSNNI.m per calcolare le transizioni da disabilitare.

Infine, la nuova rete ottenuta sarà posta in ingresso alla funzione test_SNNI.m per verificare che la rete rispetti la condizione di SNNI.

5.1 Esempio 1

Si consideri la rete di Petri in Fig. 5.1

L'unico modo per rendere la rete SNNI è disabilitare tutte le transizioni di alto livello. Tenendo conto che le transizioni di alto livello vengono introdotte per poter gestire la rete attraverso un controllore, disattivarle tutte significa rendere nulla l'azione del controllore, e quindi inutile il risultato dell'algoritmo.

Verranno effettuate tre prove:

- $PS = \{h_1\}$
- $PS = \{h_2\}$
- $PS = \{h_1, h_2\}$

Nei primi due casi la funzione makeSNNI.m dovrà ritornare un errore, mentre nel terzo caso la funzione dovrà ritornare l'insieme $D = \{h_1, h_2\}$

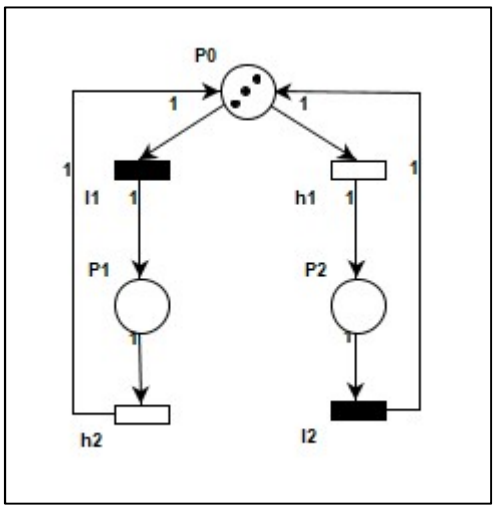

Figura 5-1 - Rete di Petri

Inserendo i seguenti dati di input

$$\boldsymbol{Pre} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \boldsymbol{Post} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{m}_0 = [3 \quad 0 \quad 0]^T$$

$$L = [1 \quad 2]$$

$$J = 5$$

nella funzione test_SNNI.m si ottiene il seguente log

```
Petri Net is not SNNI.
```

Come previsto la rete non verifica la proprietà di SNNI.

Inserendo i seguenti dati come input della funzione makeSNNI.m

$$\boldsymbol{Pre} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \boldsymbol{Post} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{m}_0 = [3 \quad 0 \quad 0]^T$$

$$L = [1 \quad 2]$$

$$J = 5$$

$$PS = [3]$$

si ottiene il seguente log

```
Error using getSNNI (line 18)
Problem does not admin solution

Error in makeSNNI (line 11)
    D = getSNNI(Pre, C, m0, L, J, PS, T, sigma_t);
```

Invece inserendo gli stessi dati di input ma cambiando l'insieme PS con

$$PS = [4]$$

si ottiene il seguente log

```
Error using getSNNI (line 18)
Problem does not admin solution

Error in makeSNNI (line 11)
    D = getSNNI(Pre, C, m0, L, J, PS, T, sigma_t);
```

Come previsto, selezionando solo una delle transizioni di alto livello non è possibile ottenere una rete SNNI.

Infine, inserendo gli stessi dati di input ma cambiando l'insieme PS con

$$PS = [3 \quad 4]$$

si ottiene il seguente log

```
ans =

     3     4
```

Come previsto, per ottenere una rete SNNI è necessario disabilitare tutte le transizioni di alto livello.

Inserendo i seguenti dati come input della funzione test_SNNI.m

$$\boldsymbol{Pre} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 0 \end{bmatrix} \boldsymbol{Post} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\boldsymbol{m}_0 = [3 \quad 0 \quad 0]^T$$

$$L = [1 \quad 2]$$

$$J = 5$$

si ottiene il seguente log

```
Petri Net is SNNI.
```

5.2 Esempio 2

Si considera la rete di Petri in Fig. 5.2

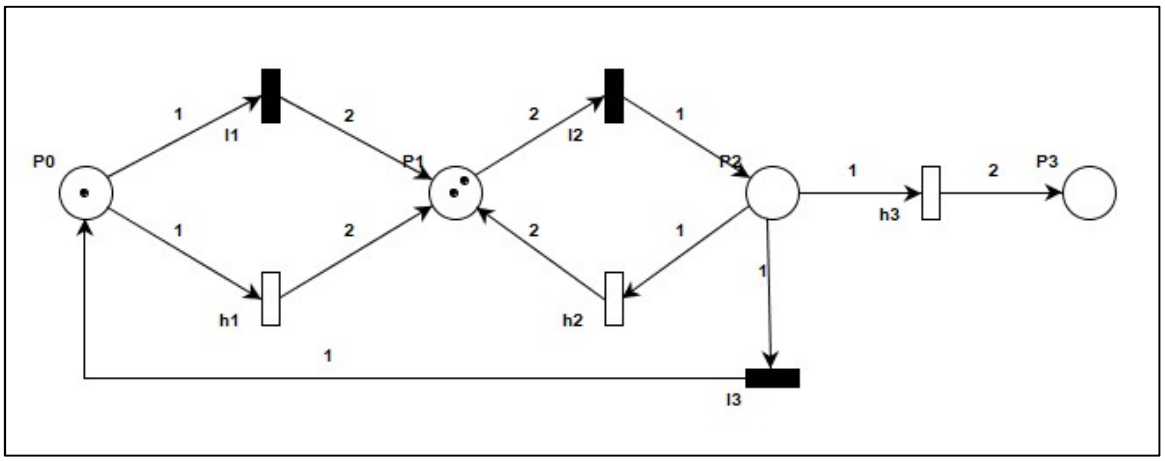

Figura 5-2 Rete di Petri

La rete non verifica la proprietà di SNNI in quanto le transizioni $H = \{h_1, h_2\}$ abilitano le transizioni di basso livello $L = \{l_1, l_2\}$.

Se si considera come insieme di transizioni che possono essere disabilitate l'insieme $PS = \{h_1, h_3\}$, la funzione makeSNNI.m dovrà restituire un errore in quanto anche disabilitando tutte le transizioni di PS la rete non potrà diventare SNNI.

Inserendo i seguenti dati in ingresso alla funzione test_SNNI.m

$$\boldsymbol{Pre} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 2 & 2 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix} \boldsymbol{Post} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{m}_0 = [1 \quad 2 \quad 0 \quad 0]^T$$

$$L = [1 \quad 2 \quad 3]$$

$$J = 5$$

La funzione produce il seguente log

```
Petri Net is not SNNI.
```

Come previsto la rete non verifica la proprietà di SNNI.

Inserendo i seguenti dati in ingresso alla funzione makeSNNI.m

$$\boldsymbol{Pre} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 2 & 0 & 0 & 2 & 2 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix} \boldsymbol{Post} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{m}_0 = [1 \quad 2 \quad 0 \quad 0]^T$$

$$L = [1 \quad 2 \quad 3]$$

$$J = 5$$

$$PS = [4 \quad 6]$$

La funzione produce il seguente log

```
Error using getSNNI (line 13)
Problem does not admin solution

Error in makeSNNI (line 11)
    D = getSNNI(Pre, C, m0, L, J, PS, T, sigma_t);
```

Come previsto, la funzione ritorna un errore.

Invece inserendo gli stessi dati di input ma cambiando l'insieme PS con

$$PS = [4 \quad 5 \quad 6]$$

Si ottiene il seguente risultato

```
ans =

     4     5
```

Ossia la rete può diventare SNNI se disabilitate le transizioni h_1 e h_2. Infatti, com'è possibile notare in Figura 5.3, disabilitando le due transizioni indicate dalla funzione

nessuna transizione di alto livello abilita un ulteriore scatto di una transizione di basso livello.

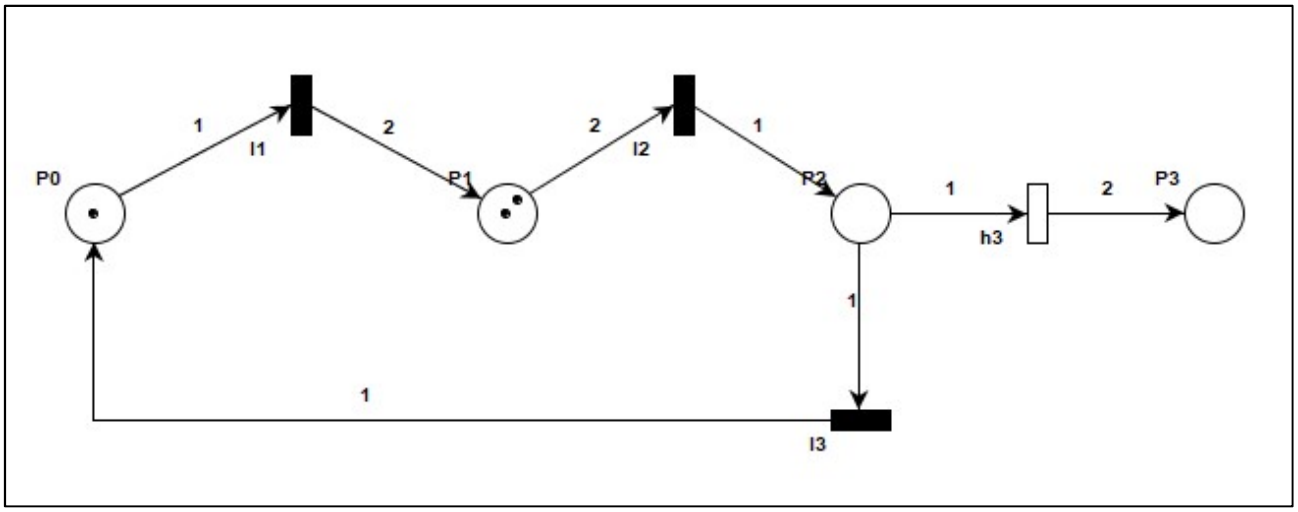

Figura 5-3 Rete di Petri della figura 5.1 senza le transizioni h_1 e h_2

Inserendo i seguenti dati in ingresso alla funzione test_SNNI.m

$$\boldsymbol{Pre} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \quad \boldsymbol{Post} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{m}_0 = [1 \quad 2 \quad 0 \quad 0]^T$$

$$L = [1 \quad 2 \quad 3]$$

$$J = 5$$

Si ottiene il seguente log

```
Petri Net is SNNI.
```

6 Conclusioni

Nel capitolo 5 sono stati riportati due casi d'uso dell'algoritmo, evidenziando come una scelta sbagliata dell'insieme delle transizioni di alto livello che possno essere disabilitate PS può rendere i problemi [3.9] – [3.10] e [3.11] – [3.12] non risolvibili.

Esattamente, se PS non contiene tutte le transizioni di alto livello che rendono la rete di Petri non SNNI, i problemi [3.9] – [3.10] e [3.11] – [3.12] non sono risolvibili.

Invece, selezionando $PS = H$, i problemi [3.9] – [3.10] e [3.11] – [3.12] diventano sicuramente risolvibili in quanto, nel caso peggiore, per rendere la rete in ingresso SNNI basta disabilitare tutte le transizioni di alto livello, ma la soluzione trovata potrebbe rendere nulla l'azione di un eventuale controllore supervisivo progettato per gestire la rete.

Una possibile sviluppo del progetto è lo sviluppo di un algoritmo che, data una rete di Petri in ingresso e l'insieme PS, venga calcolato l'insieme minimo di transizione di alto livello da disabilitare per rendere la rete non solo *SNNI*, ma in particolare *Bisimulation Strong Non deterministic Non-interference* (BSNNI), ossia una rete in cui nessuna transizione di alto livello possa essere disabilitata da una transizione di basso livello.

7 Bibliografia

[1] A. Di Febbraro e A. Giua, Sistemi ad eventi discreti, Milano: McGraw-Hill Education, 2001.

[2] F. Basile, G. De Tommasi e C. Sterle, «Non-interference enforcement in bounded Petri nets», in *57th IEEE Conference on Decision and Control (CDC'18)*, Miami Beach, Florida, December 2018.

[3] «MATLAB», [Online]. Available: https://it.wikipedia.org/wiki/MATLAB.

[4] «GLPK», [Online]. Available: https://it.wikipedia.org/wiki/GLPK.

Alla fine della tesi non possono mancare i ringraziamenti.

Ed eccomi qui, al termine di questa lunga traversata. È strano guardarsi alle spalle e rendersi conto di quanta strada è stata fatta e quante difficoltà sono state superate.

Prima di tutto devo ringraziare il mio relatore, il professor Gianmaria De Tommasi. Senza la sua guida e la grande disponibilità dimostratami questa tesi non sarebbe potuta nascere.

Ringrazio la mia famiglia per avermi supportato in ogni momento, sia nelle spese economiche sia durante le difficoltà emotive affrontate durante questo percorso.

Ringrazio la mia compagna Simona per avermi supportato, ma soprattutto sopportato, emotivamente durante questo viaggio, essendoci stata durante i momenti migliori, ma soprattutto nei momenti di maggior depressione.

Ringrazio i miei amici universitari che mi hanno accompagnato in questo viaggio con scherno e saggi consigli. Dai ragazzi che se ce l'ho fatta io ce la possono fare tutti.

Ringrazio mio zio Domenico per avermi introdotto, probabilmente inconsciamente, a questo mondo fatto di bit e bug. Se oggi sono un nerd è colpa/merito suo.

Ma soprattutto ringrazio mio nonno per avermi sempre supportato incondizionatamente, come solo un nonno sa fare. Non ha mai giudicato le mie scelte e mi ha sempre supportato nel proseguirle.

Grazie nonno baffone.