Università degli Studi di Milano Bicocca

Scuola di Scienze

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di laurea in Informatica

Conformance Checking su reti di Petri con debiti: Replay Cumulativo

Relatore: *Lucia Pomello*

Co-relatore: *Luca Bernardinello*

Relazione della prova finale di:

Andrea Premate

Matricola 829777

Anno Accademico 2019-2020

Indice

Capitolo 1

Introduzione

Il tema principale di questo elaborato è un aspetto particolare del *process mining*, ossia il *conformance checking*. A cosa ci si riferisce quando si utilizzano queste due espressioni? Il process mining si occupa in generale di tecniche che permettono l'analisi di processi di business tramite lo studio dei log di eventi, insiemi di record derivanti dalla registrazione di determinate azioni facenti parte di un processo. Ad esempio, un processo può essere la gestione di una biblioteca e le singole azioni registrate, ossia gli eventi, possono essere la richiesta di disponibilità di un libro, il ritiro di un libro, la mancata restituzione di un libro nei termini previsti e così via. Tutte queste serie di azioni vengono fatte secondo una certa logica: certamente, a meno di errori, non si registrerà mai la mancata restituzione di un libro se prima tale libro non è stato prestato. Tale logica quindi definisce un modello di processo. Il conformance checking si occupa, dato un modello di processo e un log di eventi (entrambi relativi allo stesso processo), di stabilire quanto queste due entità abbiano delle discrepanze tra di loro o dei punti in comune. La grandezza che indica quanto il log di eventi è spiegabile tramite il modello si chiama *fitness* ed assume un valore compreso tra 0 e 1, dove 0 indica che non c'è nessuna compatibilità ed 1 indica che la compatibilità è perfetta.

Abbiamo parlato però di modello senza ben definire come esso possa essere rappresentato: in questo lavoro saranno utilizzate come modello le reti di Petri.

Nella sezione iniziale infatti viene presentato formalmente questo modello matematico utilizzato generalmente nell'ambito della concorrenza. Esistono diverse tipologie di reti di Petri ma noi ne utilizzeremo alcune in particolare: le reti P/T (reti posti e transizioni). Si può pensare alle reti P/T come ad un grafo orientato con due tipi di nodi: i posti e le transizioni. I posti rappresentano gli stati locali del sistema e sono rappresentati graficamente come dei cerchi, e le transizioni rappresentano gli eventi e sono rappresentate graficamente come dei quadrati. I posti possono contenere delle marche, rappresentate graficamente con dei pallini, le quali, rispettando una certa regola di scatto definita per le reti P/T, si possono spostare di posto in posto a seconda di come le transizioni legano questi posti tra loro. Lo stato "globale" del sistema o del processo modellato è determinato dal numero di marche presenti in ciascuno dei posti della rete. Un particolare tipo di rete P/T sono le reti WF (Workflow), che, per la loro conformazione risultano essere le più adatte a descrivere il ciclo di vita di un certo processo di business: esse prevedono infatti un posto iniziale, il quale rappresenta una ipotetica situazione iniziale, ed un posto finale, che indica la fine del processo. Tra questi due particolari posti si ramificano tutti i possibili eventi che caratterizzano un processo.

Le reti P/T con debiti, che vengono presentate nella sezione 3, sono una generalizzazione delle reti P/T classiche. Esse infatti modificano la regola di scatto per le marche, rimuovendo alcuni vincoli, ed introducono la possibilità di avere un numero negativo di marche nei posti (i debiti). Questa particolare caratteristica è il presupposto sul quale si basa lo sviluppo di un metodo personale di conformance checking presentato nella sezione 5. Infatti la formazione di debiti su questo nuovo modello che ho introdotto è direttamente correlata alla possibilità di esprimere in modo più articolato la corrispondenza tra log e modello; senza tale possibilità appunto, le tracce, ossia l'insieme di eventi che si riferiscono ad uno stesso caso registrato nel log, potrebbero essere divise solamente in *compatibili* o *non compatibili* con il modello, mentre grazie alle reti di Petri con debiti è possibile definire dei gradi intermedi.

Nella sezione 4 viene presentato il conformance checking ed alcuni suoi me-

todi per calcolare il valore di fitness, ossia un valore che rappresenta quanto un modello sia conforme ad un certo log di eventi. In particolare i metodi presentati sono il Token Replay e l'Alignments. Il primo è strettamente dipendente dal modello delle reti P/T ed ha il vantaggio di essere computazionalmente snello. Il secondo ha il vantaggio di poter essere riprodotto su differenti tipi di modelli ma è computazionalmente più laborioso poiché prevede la ricerca di ottimi su un grafo.

Nella sezione 5 si presenta un metodo che ho sviluppato personalmente chiamato Replay Cumulativo. Tale metodo riprende l'idea principale del Token Replay ma, a differenza di questo metodo, conferisce una maggiore importanza al concetto di tempo, espresso in termini di scatti di transizioni. Infatti, se sul modello viene a crearsi una situazione di debito, può essere importante tener conto di quanto questa condizione duri nel tempo, prima di essere, ad esempio, "saldata". Il Replay cumulativo tiene conto di questo genere di situazioni ed è "cumulativo" proprio in questo senso: ogni comportamento considerato di interesse che si manifesta allo scatto di una transizione è importante e si va ad aggiungere (accumulare) a tutti i precedenti che saranno alla fine valutati nella loro totalità. Viene quindi fatto un confronto teorico e pratico tra il Token Replay e il Replay Cumulativo. Il confronto pratico è stato fatto su log reali ed artificiali ed è stato possibile grazie ad una precedente implementazione in Java del Replay Cumulativo e di un parser che ha consentito possibile la lettura nel programma Java dei log e delle reti. Queste ultime sono state prodotte da una funzione di process discovery (ramo del process mining che si occupa di creare modelli di processo a partire da log di eventi) della libreria Python PM4Py. Questa sezione è quella in cui viene presentato il principale contributo del lavoro di stage.

Nella sezione 6, si affronta il conformance checking con un approccio introdotto molto di recente nella letteratura e sostanzialmente differente rispetto agli altri approcci presentati: viene introdotto nella teoria un concetto di probabilità e si parla dunque di conformance checking stocastico.

Infine, nell'ultima sezione, vengono tratte le conclusioni sul lavoro svolto e

si discute di possibili sviluppi futuri.

Capitolo 2

Reti di Petri

Le reti di Petri sono dei modelli matematici utilizzati principalmente nell'ambito di sistemi distribuiti discreti. In questo lavoro le reti di Petri, in alcune particolari varianti, saranno utilizzate come modello per rappresentare processi di business. Ciò che rende questo modello utile a tale scopo è la presenza di stati locali, i quali rappresentano determinate condizioni all'interno di un processo, e la presenza di transizioni, le quali rappresentano determinate azioni che possono essere svolte dipendentemente dalle condizioni. In sostanza possiamo pensare alle reti di Petri come un grafo orientato con due tipi di nodi (stati locali e transizioni) che possono essere legati tra di loro solamente se di tipo diverso. Si parla invece di *sistema* introducendo nel modello delle reti di Petri una componente di dinamicità: le marche. Queste ultime sono elementi che si possono muovere lungo la rete, seguendo determinate regole. Lo stato "globale" di un sistema dipende dal numero e dalla posizione delle marche presenti in esso. In questo capitolo saranno presentate formalmente le reti P/T (reti posti-transizioni), le reti Workflow e altri concetti ad esse relativi [1].

2.1 Formalizzazione

Definizione 1. $N = (S, T, F, W)$ *è una* rete P/T *(rete posti-transizioni) sse:*

- (S, T, F) *è una rete, cioè*

- *S è un insieme finito di stati locali (anche chiamati posti).*
- *T è un insieme finito di transizioni tali che $S \cap T = \emptyset$ e $S \cup T \neq \emptyset$*
- *$F \subseteq (S \times T) \cup (T \times S)$ è la relazione di flusso tale che $dom(F) \cup cod(F) = S \cup T$, ossia non ci sono elementi isolati.*

- *$W : F \to \mathbb{N}$ è la funzione peso degli archi.*

Definizione 2. *$\Sigma = (S, T, F, W; M_0)$ è un* sistema P/T *(sistema posti-transizioni) sse:*

- *(S, T, F, W) è una rete P/T.*
- *$M_0 : S \to \mathbb{N}$ è la marcatura iniziale.*

Una *rete* dunque rappresenta la struttura statica su cui il *sistema* evolve a partire dalla marcatura iniziale. Ciò che determina la logica secondo cui il sistema può evolvere è la regola di scatto, che sarà definita in seguito.

Esempio 1. In seguito vi è la rappresentazione formale di un sistema P/T
$\Sigma_1 =$
$(\{p_1, p_2, p_3, p_4, p_5, p_6\},$
$\{t_1, t_2\},$
$\{(p_1, t_1), (p_2, t_1), (t_1, p_3), (p_3, t_2), (p_4, t_2), (t_2, p_6), (t_2, p_5)\},$
$\{W(p_1, t_1) = W(p_2, t_1) = W(t_1, p_3) = W(p_3, t_2) = W(t_2, p_6) = 1, W(p_4, t_2) = W(t_2, p_5) = 2\};$
$\{M_0(p_1) = 2, M_0(p_2) = 5, M_0(p_3) = 2, M_0(p_4) = 5, M_0(p_5) = 0, M_0(p_6) = 1\}).$
La figura seguente rappresenta graficamente il sistema P/T appena descritto.

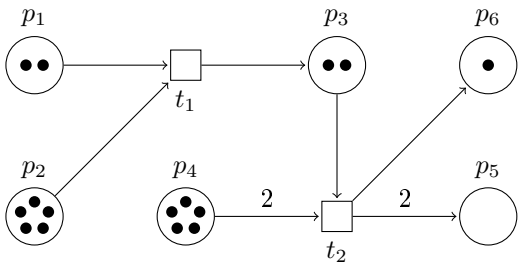


Figura 2.1: Rappresentazione del sistema Σ_1

Si noti che i posti vengono rappresentati tramite un cerchio, le transizioni con dei quadrati e la marcatura con delle marche all'interno di un posto.

Definizione 3. *Sia $X = S \cup T$ e sia $x \in X$. Chiamiamo $\bullet x = \{y \in X | (y, x) \in F\}$ i pre-elementi di x e $x\bullet = \{y \in X | (x, y) \in F\}$ i post-elementi di x.*

Esempio 2. Nel sistema Σ_1 preso a modello precedentemente possiamo ad esempio riconoscere i pre-elementi di t_1 come $\bullet t_1 = \{p_1, p_2\}$ e i post-elementi di p_4 come $p_4\bullet = \{t_2\}$.

La regola di scatto, detta anche regola di transizione, definisce invece la dinamica di un sistema P/T.

Definizione 4. *Sia $M : S \to \mathbb{N}$ una marcatura e $t \in T$. La regola di transizione è così definita:*

- *$M[t\rangle$ (t è abilitata nella marcatura M) sse $\forall s \in S$,*
  $M(s) \geq W(s, t)$
- *$M[t\rangle M'$ (dove M' è la marcatura dopo lo scatto di t) sse $\forall s \in S$,*
  $M[t\rangle \wedge\ M'(s) = M(s) - W(s, t) + W(t, s)$

Sia $\Sigma = (S, T, F, W; M_0)$ un sistema P/T:

Definizione 5. *L'insieme delle marcature raggiungibili di Σ, $[M_0\rangle$, è il più piccolo insieme tale che:*

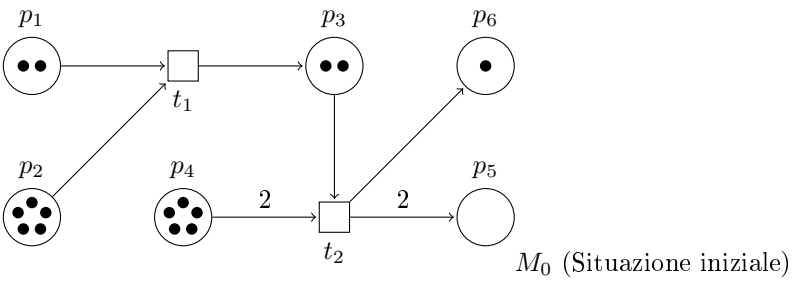
- *$M_0 \in [M_0\rangle$*
- *$M \in [M_0\rangle \wedge \exists t \in T : M[t\rangle M' \Rightarrow M' \in [M_0\rangle$*

Definizione 6. *Un multiset $U : T \to \mathbb{N}$ si dice concorrentemente abilitato in $M \in [M_0\rangle$ (U è un passo: $M[U\rangle$) e può occorrere generando M' (ossia $M[U\rangle M'$) sse $\forall s \in S$:*

- *$\sum_{t \in T} U(t) \cdot W(s, t) \leq M(s)$ (concorrentemente abilitato)*
- *$M'(s) = M(s) - \sum_{t \in T} U(t) \cdot W(s, t) + \sum_{t \in T} U(t) \cdot W(t, s)$*

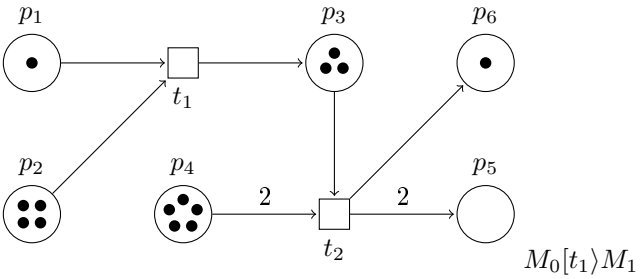
Definizione 7. *U_Σ è l'insieme dei passi di Σ tale che: $U_\Sigma = \{U : T \to \mathbb{N} | \exists M, M' \in [M_0\rangle : M[U\rangle M'\}$*

Esempio 3. Mostriamo ora qualche esempio di transizione sul sistema dell'esempio 1.



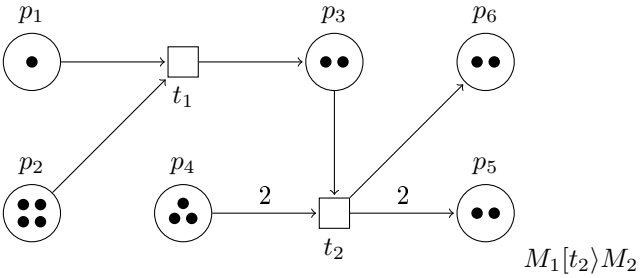
M_0 (Situazione iniziale)

Scatta t_1



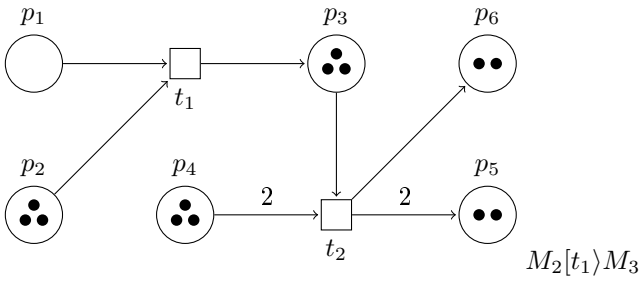
$M_0[t_1\rangle M_1$

Scatta t_2



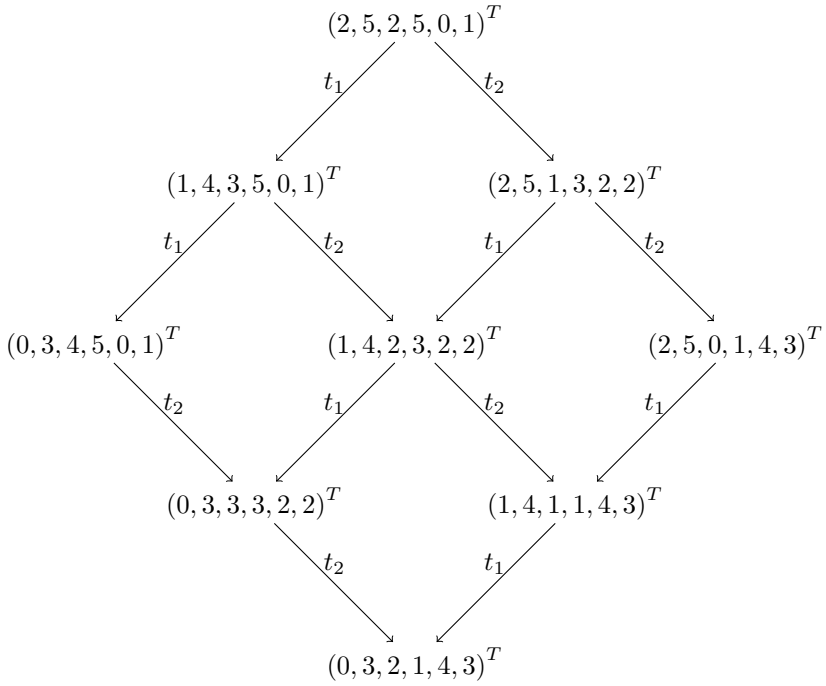
$M_1[t_2\rangle M_2$

Scatta t_1



Definizione 8. *Il grafo di raggiungibilità di* Σ *è definito come* $RG(\Sigma) = ([M_0\rangle, U_\Sigma, A, M_0)$, *dove* $A = \{(M, U, M') : M, M' \in [M_0\rangle \wedge U \in U_\Sigma \wedge M[U\rangle M'\}$. *Se, anziché* U, *si considerano solamente transizioni singole, si ha il grafo di raggiungibilità sequenziale* $SRG(\Sigma)$.

Esempio 4. Il grafo di raggiungibilità sequenziale del sistema dell'esempio 1 è così rappresentato, dove $(a, b, c, d, e, f)^T$ rappresenta il numero di marche rispettivamente nei posti $p_1, p_2, p_3, p_4, p_5, p_6$.



2.2 Rappresentazione algebrica

È possibile inoltre rappresentare un sistema P/T $\Sigma = (S, T, F, W; M_0)$ in modo algebrico, con un interessante vantaggio per lo studio di molte proprietà.

Definizione 9. *Σ può essere rappresentato da due matrici con $|S|$ righe e $|T|$ colonne e un vettore m_0. Chiamiamo matrice backward la matrice $Pre \in \mathbb{N}^{|S|\times|T|}$ dove $Pre[i,j] = W(s_i, t_j)$ e chiamiamo matrice forward la matrice $Post \in \mathbb{N}^{|S|\times|T|}$ dove $Post[i,j] = W(t_j, s_i)$. Chiamiamo matrice di incidenza la matrice $C = Post - Pre$, se $N = (S, T, F)$ è senza cappi($\nexists s \in S : s \in (s\bullet)\bullet$).*

La marcatura iniziale M_0 è rappresentata da un vettore colonna $m_0 \in \mathbb{N}^{|S|}$.

Esempio 5. In seguito vi è la rappresentazione della matrice backward, della matrice forward, della matrice di incidenza e del vettore della marcatura iniziale del sistema Σ_1.

Pre	t_1	t_2
p_1	1	0
p_2	1	0
p_3	0	1
p_4	0	2
p_5	0	0
p_6	0	0

$Post$	t_1	t_2
p_1	0	0
p_2	0	0
p_3	1	0
p_4	0	0
p_5	0	2
p_6	0	1

C	t_1	t_2
p_1	-1	0
p_2	-1	0
p_3	1	-1
p_4	0	-2
p_5	0	2
p_6	0	1

$$m_0 = \begin{bmatrix} 2 \\ 5 \\ 2 \\ 5 \\ 0 \\ 1 \end{bmatrix}$$

Definizione 10. *La regola di scatto può essere definita nel seguente modo: sia il vettore* $m \in \mathbb{N}^{|S|}$ *una marcatura e* $t \in T$

- $m[t\rangle$ *(t è abilitata nella marcatura m) sse* $m \geq Pre[\bullet, t]$
- $m[t\rangle m'$ *(dove* m' *è la marcatura dopo lo scatto di t) sse* $m[t\rangle \wedge\ m' = m + Post[\bullet, t] - Pre[\bullet, t] = m + C[\bullet, t]$

dove Pre$[\bullet, t]$ rappresenta il vettore colonna (Pre$[s_1, t]$, Pre$[s_2, t]$, ...,Pre$[s_{|S|}, t]$) della matrice Pre e in modo analogo sono definiti Post$[\bullet, t]$ e $C[\bullet, t]$.

Queste definizioni sono state date prendendo a modello le definizioni di [2] riguardanti le reti P/T.

Definizione 11. *Sia* σ *una sequenza di transizioni, definiamo vettore di Parikh di* σ *il vettore colonna* c_σ *di* $|T|$ *elementi tale che* $c_\sigma(t_i)$ *è il numero di occorrenze di* t_i *in* σ.

Definizione 12. *Sia* c_σ *un vettore di Parikh,* m_0 *la marcatura iniziale e* C *la matrice di incidenza di un certo sistema* Σ. *Definiamo equazione di stato:* $m_0 + C \cdot c_\sigma = m_1$

Nelle reti P/T, affinché la marcatura prodotta dall'equazione di stato sia una marcatura effettivamente raggiungibile, è necessario che la sequenza σ sia abilitata, ossia $m_0[\sigma\rangle m_1$.

Esempio 6. Mostriamo come calcolare direttamente, tramite l'equazione di stato, la marcatura finale che abbiamo mostrato nell'esempio 3. Andando a contare quindi il numero delle diverse transizioni t_i scattate troviamo: t_1 si è presentata 2 volte, t_2 si è presentata 1 volta. Il vettore di Parikh quindi sarà $c_\sigma = \begin{bmatrix} 2 & 1 \end{bmatrix}^T$.

$$\text{Quindi } m_3 = \begin{bmatrix} 2 \\ 5 \\ 2 \\ 5 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} -1 & 0 \\ -1 & 0 \\ 1 & -1 \\ 0 & -2 \\ 0 & 2 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 3 \\ 3 \\ 2 \\ 2 \end{bmatrix}$$

ottenendo così esattamente la stessa marcatura finale che avevamo raggiunto.

2.3 Reti Workflow

Introduciamo ora le reti WF (workflow) [3] poichè sono le più adatte a descrivere il ciclo di vita di un certo processo di business. Pensiamo ad esempio all'acquisto online di uno o più prodotti: l'insieme di attività che possiamo svolgere è limitato e dipende dalle attività svolte in precedenza. Se ad esempio abbiamo aggiunto un prodotto nel carrello allora potremo andare a pagare oppure decidere di aggiungerne un altro; se non abbiamo aggiunto nulla nel carrello non potremo decidere di andare a pagare; se abbiamo diversi oggetti nel carrello e cambiamo idea sull'acquisto di uno di essi, allora potremo decidere di rimuoverlo dal carrello.

Definizione 13. *Sia $N = (S, T, F)$ una rete P/T (consideriamo ogni arco avente peso 1 e quindi non introduciamo la funzione W) e sia $\bar{t} \notin (S \cup T)$. N è una rete workflow sse:*

- *S contiene un posto di input i tale che $\bullet i = \emptyset$*
- *S contiene un posto di output o tale che $o\bullet = \emptyset$*
- *$\overline{\mathrm{N}} = (S, T \cup \{\bar{t}\}, F \cup \{(o, \bar{t}), (\bar{t}, i)\})$ è fortemente connessa, ossia esiste un cammino tra ogni coppia di nodi in $\overline{\mathrm{N}}$.*

Esempio 7. Sia $N = (S, T, F)$ una rete workflow. La figura seguente è la rappresentazione di $\overline{\mathrm{N}} = (S, T \cup \{\bar{t}\}, F \cup \{(o, \bar{t}), (\bar{t}, i)\})$.

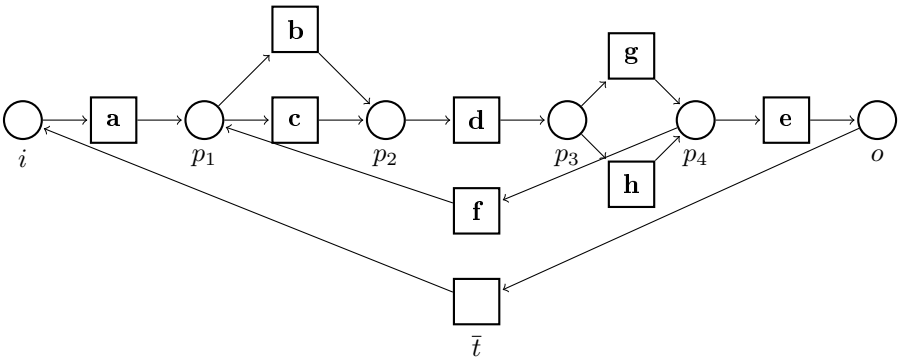


Figura 2.2: Rappresentazione di $\overline{\mathrm{N}}$

Come si può osservare dalla figura, la transizione $\bar{t}$ serve a collegare lo stato di output o con lo stato di input i della rete, rendendo così la rete fortemente connessa.

Capitolo 3

Reti di Petri con debiti

Il motivo per cui introduciamo le reti di Petri con debiti è quello di poter utilizzare la teoria delle reti di Petri tradizionali, generalizzandola in modo da renderla utile alla nostra applicazione riguardante il conformance checking: in particolare nelle reti di Petri con debiti non abbiamo alcun vincolo che possa impedire lo scatto di una transizione e ciò è reso possibile ammettendo delle marche "negative". In seguito quindi viene rappresentata la formalizzazione di questo tipo di reti, andando a descrivere ciò che le differenzia dalle reti di Petri tradizionali.

3.1 Formalizzazione e rappresentazione algebrica

Definizione 14. $N = (S, T, F, W)$ *è una* rete P/T con debiti *sse:*

- (S, T, F) *è una rete, cioè*
  - S *è un insieme finito di stati locali.*
  - T *è un insieme finito di transizioni tali che* $S \cap T = \emptyset$ *e* $S \cup T \neq \emptyset$
  - $F \subseteq (S \times T) \cup (T \times S)$ *è la relazione di flusso tale che* $dom(F) \cup cod(F) = S \cup T$*, ossia non ci sono elementi isolati.*
- $W : F \to \mathbb{N}$ *è la funzione peso degli archi.*

Definizione 15. $\Sigma = (S, T, F, W; M_0)$ *è un* sistema P/T con debiti *sse:*

- (S, T, F, W) *è una rete P/T con debiti.*
- $M_0 : S \to \mathbb{N}$ *è la marcatura iniziale.*

Si noti che la definizione formale dei sistemi P/T con debiti è identica alla definizione dei sistemi P/T tradizionali. Ciò che cambia infatti è nella regola di scatto: sarà consentito raggiungere negli stati locali un numero di marche negativo. È stato scelto di definire la marcatura iniziale allo stesso modo dei sistemi P/T tradizionali e non ammettendo anche marche negative, poiché la si considera come un momento iniziale in cui ancora ci sono delle condizioni di ordinarietà.

Definizione 16. *Sia $M : S \to \mathbb{Z}$ una marcatura e $t \in T$. La regola di transizione è così definita:*

- $M[t\rangle$ *è sempre abilitata.*
- $M[t\rangle M'$ *dove M' è la marcatura dopo lo scatto di t*
  $M'(s) = M(s) - W(s,t) + W(t,s), \forall s \in S$

Oppure algebricamente:

Definizione 17. *La regola di scatto può essere definita nel seguente modo: sia il vettore $m \in \mathbb{Z}^{|S|}$ una marcatura e $t \in T$*

- $m[t\rangle$ *è abilitata sempre.*
- $m[t\rangle m'$
  $m' = m + Post[\bullet, t] - Pre[\bullet, t] = m + C[\bullet, t]$

dove Pre$[\bullet, t]$ rappresenta il vettore colonna (Pre$[s_1, t]$, Pre$[s_2, t]$, ...,Pre$[s_{|S|}, t]$) della matrice Pre e in modo analogo sono definiti Post$[\bullet, t]$ e $C[\bullet, t]$.

Queste definizioni sono state date prendendo a modello le definizioni di [2] riguardanti le reti P/T e adattate al caso di reti P/T con debiti.

Tutte le altre definizioni che sono state date nella sezione precedente possono essere direttamente applicate al caso delle reti di Petri con debiti. In seguito

quindi vengono mostrati alcuni esempi analoghi a quelli della sezione precedente, adattati alle reti di Petri con debiti. In particolare sarà presa in considerazione la stessa rete ma con una marcatura iniziale differente.

Esempio 8. In seguito vi è la rappresentazione formale di un sistema P/T con debiti $\Sigma_2 =$
$(\{p_1, p_2, p_3, p_4, p_5, p_6\},$
$\{t_1, t_2\},$
$\{(p_1, t_1), (p_2, t_1), (t_1, p_3), (p_3, t_2), (p_4, t_2), (t_2, p_6), (t_2, p_5)\},$
$\{W(p_1, t_1) = W(p_2, t_1) = W(t_1, p_3) = W(p_3, t_2) = W(t_2, p_6) = 1, W(p_4, t_2) = W(t_2, p_5) = 2\};$
$\{M_0(p_1) = 0, M_0(p_2) = 5, M_0(p_3) = 0, M_0(p_4) = 1, M_0(p_5) = 0, M_0(p_6) = 1\}).$
La figura seguente rappresenta graficamente il sistema P/T con debiti appena descritto.

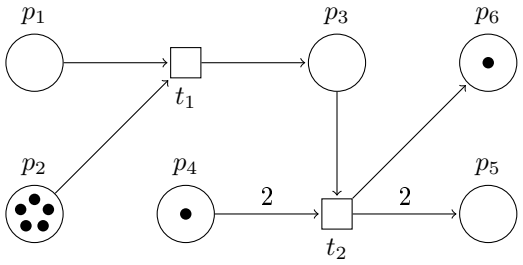
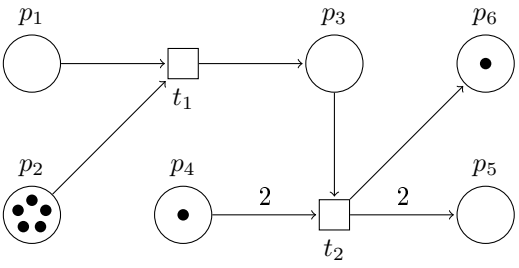


Figura 3.1: Rappresentazione del sistema Σ_2

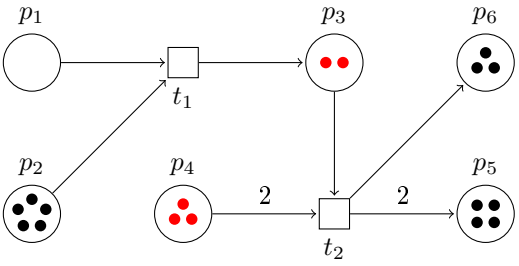
Per rappresentare le marche presenti in un posto in numero positivo saranno utilizzati dei pallini neri, mentre per rappresentare le marche presenti in un posto in numero negativo saranno utilizzati dei pallini rossi.

Esempio 9. Mostriamo ora qualche esempio di transizione sul sistema Σ_2.



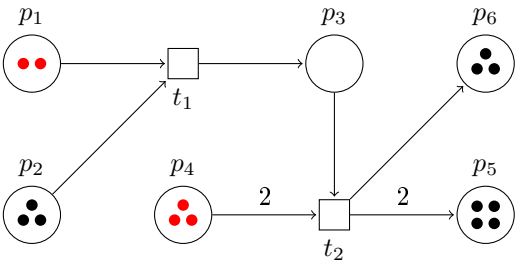
M_0 (Situazione iniziale)

scatta due volte t_2



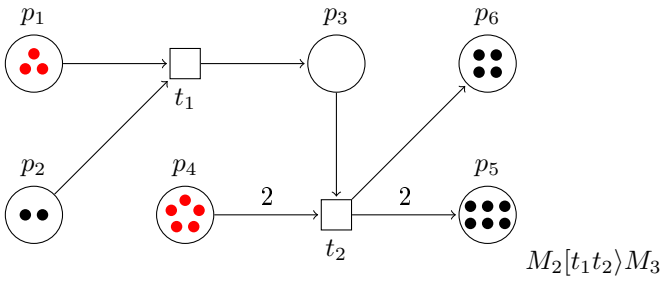
$M_0[t_2t_2\rangle M_1$

Scatta due volte t_1



$M_1[t_1t_1\rangle M_2$

Scattano t_1 e t_2



Esempio 10. In seguito vi è la rappresentazione della matrice backward, della matrice forward, della matrice di incidenza e del vettore della marcatura iniziale del sistema Σ_2.

Pre	t_1	t_2
p_1	1	0
p_2	1	0
p_3	0	1
p_4	0	2
p_5	0	0
p_6	0	0

$Post$	t_1	t_2
p_1	0	0
p_2	0	0
p_3	1	0
p_4	0	0
p_5	0	2
p_6	0	1

C	t_1	t_2
p_1	-1	0
p_2	-1	0
p_3	1	-1
p_4	0	-2
p_5	0	2
p_6	0	1

$$m_0 = \begin{bmatrix} 0 \\ 5 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

Si noti che la matrice backward, la matrice forward e la matrice di incidenza sono identiche a quelle dell'esempio della sezione precedente (Σ_1), poiché la rete è la medesima. Ciò che cambia è invece la marcatura iniziale.

Esempio 11. Mostriamo ora come calcolare direttamente, tramite l'equazione di stato, la marcatura finale. Andando a contare quindi il numero delle diverse transizioni t_i scattate troviamo: t_1 si è presentata 3 volte, t_2 si è presentata 3 volte. Il vettore di Parikh quindi sarà $c_\sigma = \begin{bmatrix} 3 & 3 \end{bmatrix}^T$.

$$\text{Quindi } m_3 = \begin{bmatrix} 0 \\ 5 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} -1 & 0 \\ -1 & 0 \\ 1 & -1 \\ 0 & -2 \\ 0 & 2 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} -3 \\ 2 \\ 0 \\ -5 \\ 6 \\ 4 \end{bmatrix}$$

ottenendo così esattamente la stessa marcatura finale che avevamo raggiunto. Dal momento che, a differenza delle classiche reti P/T, la condizione $m[t\rangle$ è sempre soddisfatta per le reti P/T con debiti, possiamo affermare che l'equazione di stato è condizione necessaria e sufficiente affinché una sequenza di transizioni generi una marcatura in un sistema.

Esempio 12. Il grafo di raggiungibilità sequenziale del sistema Σ_2 è rappresentato in figura 3.2 e $(a, b, c, d, e, f)^T$ rappresenta il numero di marche rispettivamente nei posti $p_1, p_2, p_3, p_4, p_5, p_6$.

Si noti che il grafo di raggiungibilità di un sistema P/T con debiti è sempre un grafo infinito dal momento che, a differenza dei classici sistemi P/T, ogni transizione è abilitata in ogni momento.

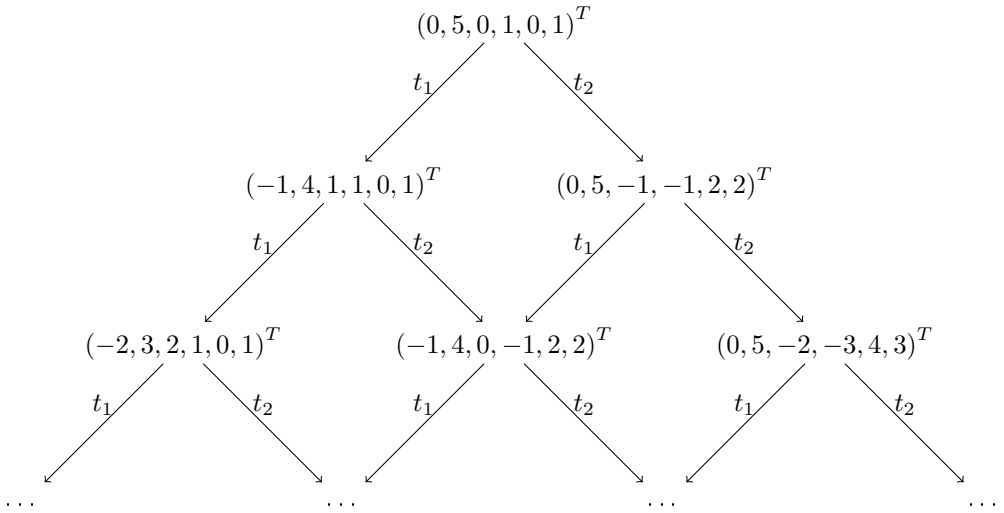


Figura 3.2: Grafo di raggiungibilità sequenziale del sistema Σ_2

Capitolo 4

Conformance Checking

Il conformance checking è una tecnica del process mining che permette di comparare un modello di processo ed un log di eventi, entrambi relativi allo stesso processo. Il processo viene modellato dalle reti Workflow (WF). Si rimanda alla sezione sottostante, relativa ai concetti base, per una spiegazione che fornisca la giusta intuizione sulla natura delle parti chiamate in causa in questa breve introduzione. Successivamente saranno presentati due metodi di conformance checking presenti in letteratura ed un loro breve confronto. Questa intera sezione è basata sul capitolo 8 di [3].

4.1 Concetti base

Definiamo anzitutto le due principali entità che sono oggetto di analisi del conformance checking: modello di processo e log di eventi.
Il log di eventi è un insieme di eventi che vengono registrati, ad esempio da un sistema informatico, e che fanno riferimento a certe attività. Per rendere meglio l'idea riportiamo un frammento della tabella 2.1 di [3].

Case id	Event id	Timestamp	Activity	Resource	Cost	...
1	35654423	30-12-2010:11.02	register request	Pete	50	...
	35654424	31-12-2010:10.06	examine thoroughly	Sue	400	...
	35654425	05-01-2011:15.12	check ticket	Mike	100	...
	35654426	06-01-2011:11.18	decide	Sara	200	...
	35654427	07-01-2011:14.24	reject request	Pete	200	...
2	35654483	30-12-2010:11.32	register request	Mike	50	...
	35654485	30-12-2010:12.12	check ticket	Mike	100	...
	35654487	30-12-2010:14.16	examine casually	Pete	400	...
	35654488	05-01-2011:11.22	decide	Sara	200	...
	35654489	08-01-2011:12.05	pay compensation	Ellen	200	...
3	35654521	30-12-2010:14.32	register request	Pete	50	...
	35654522	30-12-2010:15.06	examine casually	Mike	400	...
	35654524	30-12-2010:16.34	check ticket	Ellen	100	...
	35654525	06-01-2011:09.18	decide	Sara	200	...
	35654526	06-01-2011:12.18	reinitiate request	Sara	200	...
	35654527	06-01-2011:13.06	examine thoroughly	Sean	400	...
...	...	...	...	...	...	...

Figura 4.1: Esempio di log di eventi.

Come vediamo, può essere considerato una sorta di resoconto tabellare di un insieme di attività (ciascuna contenente degli attributi), differenziate per caso: ognuna delle attività presenti fa parte di un certo compito o *task* che viene svolto, rappresentato in tabella dalla colonna -Case id-. In seguito un evento sarà indicato con una lettera minuscola, mentre la sequenza di eventi (Case id) verrà chiamata traccia e indicata con σ.
Il modello è invece tipicamente un grafo, in particolare noi utilizzeremo le reti WF ma esistono molte altre rappresentazioni, che mette in mostra le interconnessioni tra le varie attività possibili. In particolare le transizioni delle reti WF rappresenteranno ciò che chiamiamo attività/evento. In seguito riportiamo un

esempio di modello (tratto dalla figura 8.2 di [3]) che sarà utilizzato anche nei prossimi paragrafi.

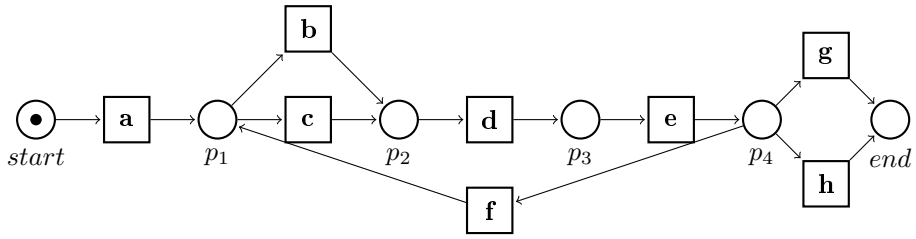


Figura 4.2: Rete WF corrispondente ad un modello di processo.

Vediamo come un modello stabilisca con una certa logica, dipendente dalla propria morfologia, le diverse possibilità di successioni di eventi. Nel modello riportato ad esempio possiamo notare come la sequenza di eventi $\langle a, b, d, e, g \rangle$ sia contemplata dal nostro modello, mentre la sequenza $\langle a, f, d, e, g \rangle$ no.
Il conformance checking si occupa principalmente di valutare la corrispondenza tra un modello di processo, il quale può essere costruito a priori o scoperto tramite algoritmi di process discovery, ed un log di eventi. Solitamente questa corrispondenza è espressa con un valore compreso tra 0 e 1 e viene chiamata *fitness*: maggiori sono i punti in comune tra modello e log di eventi più il valore di fitness si avvicinerà a 1 e, viceversa, meno punti in comune ci sono e più la fitness si avvicinerà allo 0. Cosa significa "avere punti in comune" per due entità, ossia modello e log di eventi, che sono due oggetti così differenti? Il concetto di *replay* aiuta a chiarire questo discorso: il log di eventi viene riprodotto sul modello e, quanto più è possibile riprodurre nel modello gli stessi eventi presenti nel log, tanto più si ha una elevata fitness. Si possono dare diverse definizioni di fitness, e, in base a questo, diversi saranno i valori che si otterranno per lo stesso modello, rispetto allo stesso log di eventi. Nei paragrafi successivi sono presentati diversi modi per valutare la fitness.

4.2 Token Replay

Gli esempi illustrati in seguito sono tratti dal capitolo 8.2 di [3] con alcuni adattamenti.

Immaginiamo di avere una WF-net N_1 corrispondente al nostro modello di processo e le tracce $\sigma_0 = \langle a, b, d, e, g \rangle$ e $\sigma_1 = \langle a, d, c, e, h \rangle$ provenienti dal log di eventi. Secondo il metodo che viene ora illustrato, bisogna cercare di riprodurre una traccia del log di eventi su/sui modelli dati.

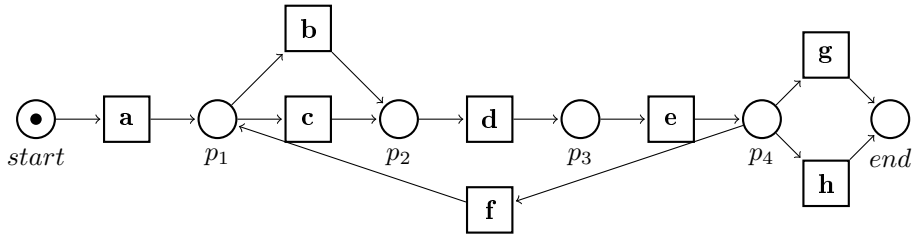
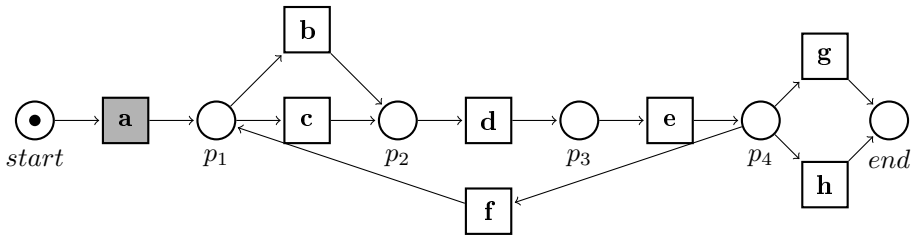
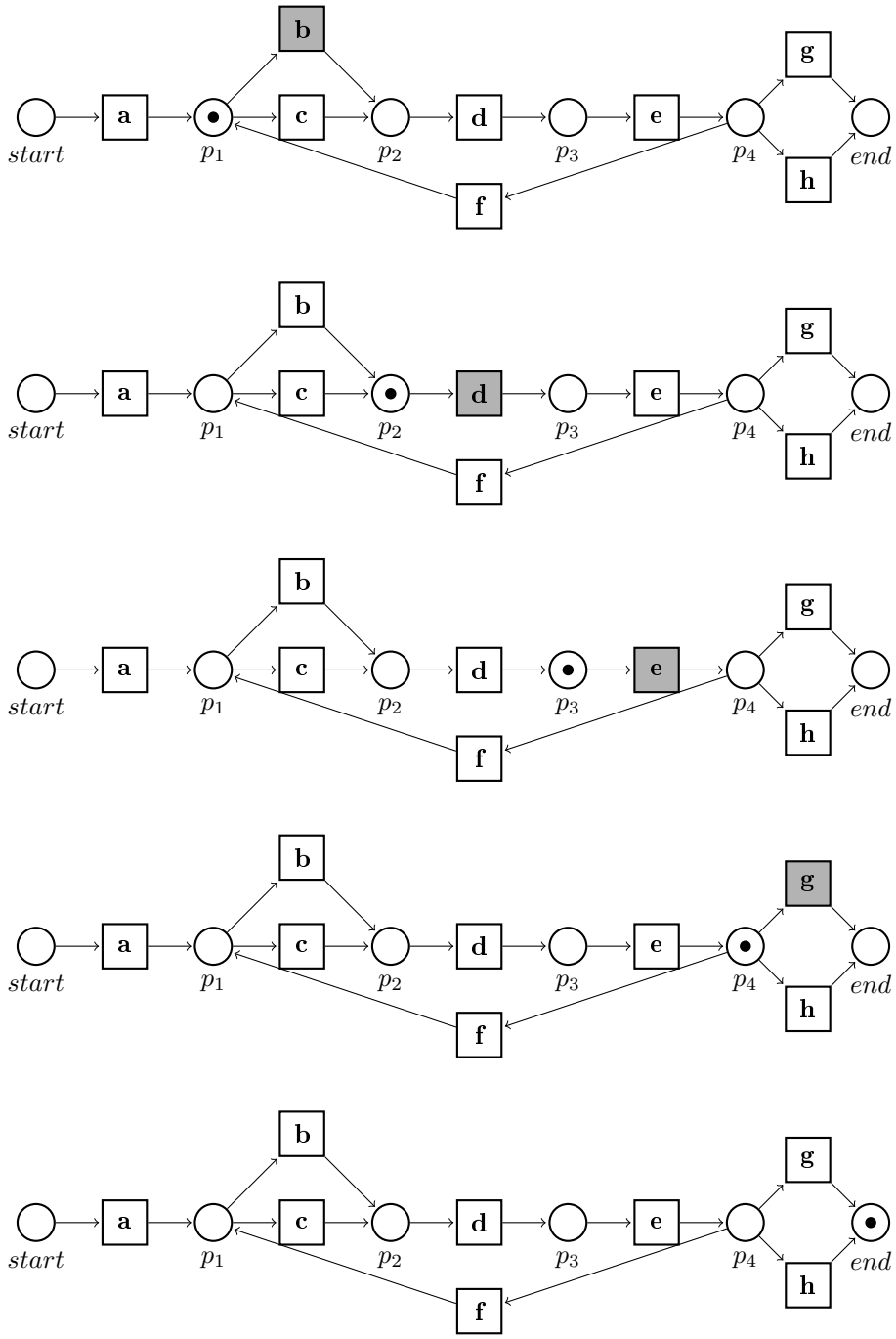


Figura 4.3: WF-net N_1 corrispondente al modello di processo.

Volendo riprodurre la traccia σ_0 sulla rete N_1 ci accorgiamo subito che questo è possibile senza alcun tipo di problema. Ecco riportato graficamente ciò che avviene.





Intuitivamente non ci sarebbe motivo quindi di penalizzare in alcun modo la fitness, che infatti sarà 1, il valore massimo.

Se invece consideriamo la traccia σ_1 e la rete N_1:

- la prima transizione a scatta senza problemi;

- la transizione d non potrebbe scattare secondo il nostro modello però consentiamo che scatti e ci segniamo che è mancata una marca ($m = 1$);

- le transizioni c, e, h scattano senza problemi e l'esecuzione termina lasciando in p_2 una marca rimanente ($r = 1$);

Se teniamo il conto anche delle marche prodotte in totale, compresa quella iniziale ($p = 6$) e di tutte quelle che abbiamo consumato durante l'esecuzione comprendendo quella finale ($c = 6$) abbiamo abbastanza informazioni per dare una definizione di fitness.

La fitness di una WF-net N e una traccia σ è definita nel seguente modo:

$$\mathit{fitness}(\sigma, N) = \tfrac{1}{2}\left(1 - \tfrac{m}{c}\right) + \tfrac{1}{2}\left(1 - \tfrac{r}{p}\right)$$

La prima parte calcola il rapporto tra le marche mancanti m e le marche consumate c, la seconda parte calcola il rapporto tra le marche rimanenti r e le marche prodotte p. Si noti che se le marche mancanti e le marche rimanenti sono 0 ci troviamo nella situazione ideale in cui la nostra traccia σ viene simulata alla perfezione dal modello N, come nel primo esempio. Al contrario, maggiore è il numero di marche mancanti e marche rimanenti (in rapporto rispettivamente con le marche consumate e quelle prodotte), più la fitness si avvicinerà a 0. Si noti che viene attribuito lo stesso peso, ossia $\frac{1}{2}$, sia alla parte delle marche mancanti che a quella delle marche rimanenti; tale decisione può essere modificata dando più peso alla caratteristica che si ritiene gravi di più sulla fitness.

Tornando quindi al nostro esempio, in base ai valori calcolati di p, c, m e r:

$$\mathit{fitness}(\sigma_1, N_1) = \tfrac{1}{2}\left(1 - \tfrac{1}{6}\right) + \tfrac{1}{2}\left(1 - \tfrac{1}{6}\right) = 0.8333$$

Fino ad ora abbiamo però calcolato la fitness di una sola traccia del log di eventi. Dal momento che un log di eventi presenta un consistente numero di tracce, ognuna con la relativa frequenza, dobbiamo trovare un modo di generalizzare il discorso fatto per una singola traccia. Sia dunque $p_{N,\sigma}$ il numero di marche prodotte quando eseguiamo la traccia σ su N e definiamo in modo del tutto analogo $c_{N,\sigma}$, $m_{N,\sigma}$ e $r_{N,\sigma}$ basandoci sulle precedenti definizioni di c, m e

r; sia $L(\sigma)$ la frequenza di una traccia σ. La fitness di un log di eventi L su una WF-net N è definita come:

$$\mathit{fitness}(L,N) = \frac{1}{2}\left(1 - \frac{\sum_{\sigma \in L} L(\sigma) \times m_{N,\sigma}}{\sum_{\sigma \in L} L(\sigma) \times c_{N,\sigma}}\right) + \frac{1}{2}\left(1 - \frac{\sum_{\sigma \in L} L(\sigma) \times r_{N,\sigma}}{\sum_{\sigma \in L} L(\sigma) \times p_{N,\sigma}}\right)$$

Se ipotizziamo quindi un atipico log di eventi contenente solo 12 tracce σ_0 e 8 tracce σ_1 (solitamente sono presenti un numero molto maggiore di tracce) possiamo immediatamente calcolare la fitness tra il log e la rete N_1.

$$\mathit{fitness}(L,N) = \frac{1}{2}\left(1 - \frac{12\times 0 + 8\times 1}{12\times 6 + 8\times 6}\right) + \frac{1}{2}\left(1 - \frac{12\times 0 + 8\times 1}{12\times 6 + 8\times 6}\right) = 0.9333$$

A questo punto ci possiamo domandare se il seguente modello non sia migliore del precedente, sempre considerando lo stesso log di eventi:

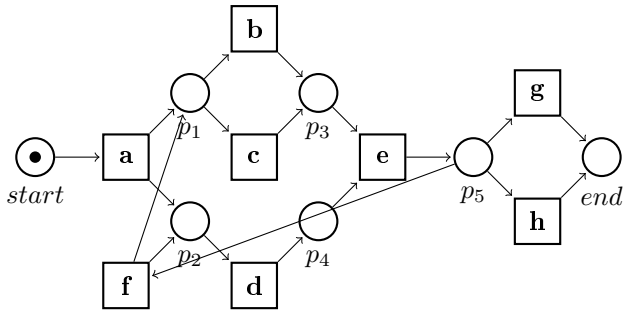


Figura 4.4: WF-net N_2 di un modello di processo alternativo.

Riproducendo $\sigma_0 = \langle a, b, d, e, g\rangle$ su N_2 abbiamo: 0 marche mancanti, 7 marche prodotte, 7 marche consumate e 0 marche rimanenti (la fitness relativa a σ_0 e N_2 è 1). Un ragionamento simile può essere fatto con $\sigma_1 = \langle a, d, c, e, h\rangle$ su N_2. La conclusione è quindi che, dal momento che la fitness tra L e N_2 è 1 > 0.9333, possiamo stabilire che il modello N_2 si adatta meglio al log di eventi L rispetto a quanto non faccia N_1. In questo caso particolare non servivano i conti per convincersene, dal momento che N_2 poteva riprodurre entrambe le tracce perfettamente mentre N_1 no, ma più in generale, quando non si ha una situazione banale come questa, diventa necessaria l'applicazione della formula.

4.3 Alignments

Gli esempi illustrati in seguito sono tratti dal capitolo 8.2 di [3] con alcuni adattamenti.

Per spiegare il concetto di Alignments informalmente consideriamo sempre le nostre tracce $\sigma_0 = \langle a, b, d, e, g \rangle$, $\sigma_1 = \langle a, d, c, e, h \rangle$ e la nostra rete N_1 del paragrafo precedente che riportiamo per comodità.

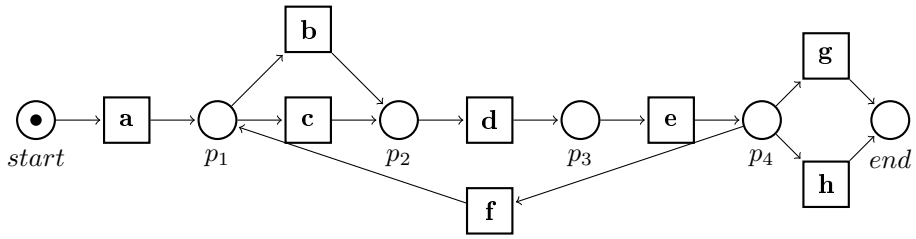


Figura 4.5: WF-net N_1 corrispondente al modello di processo.

Possiamo osservare con facilità che σ_0 si adatta perfettamente alla rete N_1:

$$\gamma_0 = \left|\begin{array}{|c|c|c|c|c|}\hline a & b & d & e & g \\ \hline a & b & d & e & g \\ \hline \end{array}\right|$$

La riga superiore corrisponde a σ_0 e la riga inferiore corrisponde al percorso su N_1 dalla marcatura iniziale a quella finale, cercando di riprodurre la traccia nel modo più preciso possibile. Se consideriamo invece σ_1 sempre sulla rete N_1 esistono diversi allineamenti ottimali, ossia gli allineamenti per cui si ha la massima corrispondenza possibile tra modello e traccia:

$$\gamma_{1a} = \left|\begin{array}{|c|c|c|c|c|c|}\hline a & \gg & d & c & e & h \\ \hline a & b & d & \gg & e & h \\ \hline \end{array}\right|, \quad \gamma_{1b} = \left|\begin{array}{|c|c|c|c|c|c|}\hline a & \gg & d & c & e & h \\ \hline a & c & d & \gg & e & h \\ \hline \end{array}\right|, \quad \gamma_{1c} = \left|\begin{array}{|c|c|c|c|c|c|}\hline a & d & c & \gg & e & h \\ \hline a & \gg & c & d & e & h \\ \hline \end{array}\right|$$

Il simbolo $\gg$ indica che c'è stato un disallineamento: quando c'è una discrepanza tra log e modello e non si può eseguire la stessa transizione allora, log o modello hanno bisogno di una mossa di "riallineamento" per tornare in assetto con la controparte. In questi semplici casi si può calcolare quasi a occhio quale sia un allineamento ottimale, ma in genere, specie con una rete ed una traccia più estesi, questo problema non è così banale. L'algoritmo che si occupa di calcolare l'allineamento ottimale è l'algoritmo A^*, che è una generalizzazione dell'algoritmo di Dijkstra [4], un algoritmo utilizzato per cercare i cammini minimi da sorgente unica in un grafo orientato pesato nel caso in cui tutti i pesi degli archi siano non negativi [5].

Cerchiamo di approfondire il collegamento tra i due. Anzitutto l'algoritmo di Dijkstra è un algoritmo per grafi "semplici" ossia grafi composti solo da vertici e archi, diversamente dalle reti di Petri. Consideriamo quindi un grafo $G = (V, E)$ dove V è l'insieme dei vertici ed E l'insieme degli archi orientati, associamo ad ogni arco un valore w, detto peso dell'arco, e stabiliamo la sorgente $s \in V$. L'algoritmo di Dijkstra calcola tutte le distanze tra s e i vertici in V tali che la somma dei pesi degli archi percorsi è la minore possibile. Come può aiutarci questo algoritmo a calcolare un allineamento ottimale? L'elemento chiave è il cosiddetto prodotto sincrono, una rete di Petri costruita essenzialmente combinando il modello di processo e la traccia dei quali vogliamo trovare l'allineamento. A questo punto viene calcolato il grafo dei casi del prodotto sincrono e trovati quindi tramite Dijkstra gli allineamenti ottimali. In particolare ciò che si vuole massimizzare è, all'interno del prodotto sincrono, il numero di mosse che sono sia del modello che della traccia.

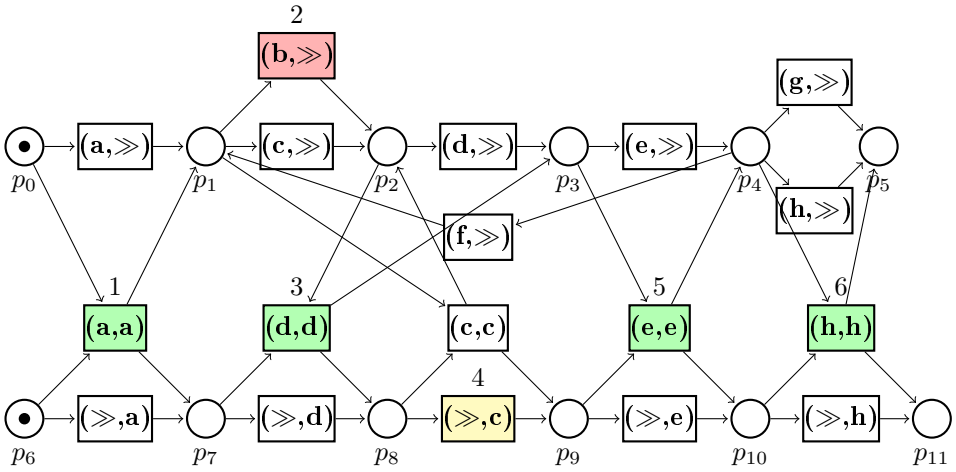


Figura 4.6: Prodotto sincrono della traccia σ_0 e del modello N_1. In evidenza sono state messe le transizioni toccate dall'allineamento ottimale γ_{1a}. In verde sono evidenziate le mosse sia del modello che del log, in rosso le mosse esclusivamente del modello ed in giallo le mosse esclusivamente del log.

Esempio 13. Nelle figure 4.7-4.11 vi è un esempio di funzionamento dell'algoritmo di Dijkstra su un grafo pesato allo scopo di dare una idea intuitiva dell'algoritmo. In particolare si vuole trovare la distanza dal nodo a, detto anche nodo sorgente, di tutti i nodi del grafo.

Per selezionare l'allineamento più appropriato dobbiamo quindi associare un costo alle mosse indesiderate e verrà così trovato un allineamento con il più basso costo totale. La funzione che si occupa di ciò è la funzione di costo δ: le mosse in cui il log e il modello corrispondono hanno costo 0 mentre possono essere dati valori diversi ai costi delle altre attività in base alla loro natura, ossia se in corrispondenza di una certa attività a c'è un disallineamento e consideriamo questo disallineamento grave possiamo ad esempio assegnare un valore $\delta(a, \gg) = 5$, mentre se la stessa cosa accade con una attività b meno importante allora gli assegneremo un valore inferiore come $\delta(b, \gg) = 2$. Un allineamento è ottimale se non ci sono altri allineamenti con costo totale minore, dove il costo totale è la somma di tutti i costi di un certo allineamento γ.

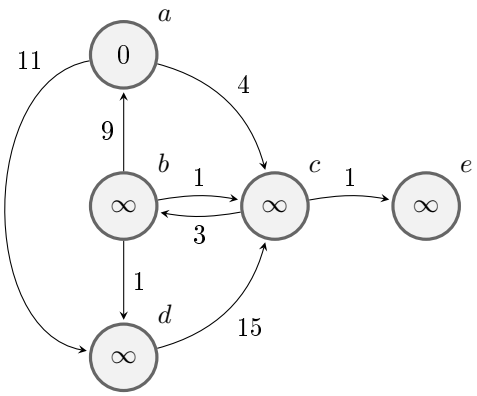


Figura 4.7: Anzitutto viene inizializzata la sorgente a distanza 0 (ogni nodo è a distanza 0 da sè stesso) e tutti gli altri nodi a distanza ∞.

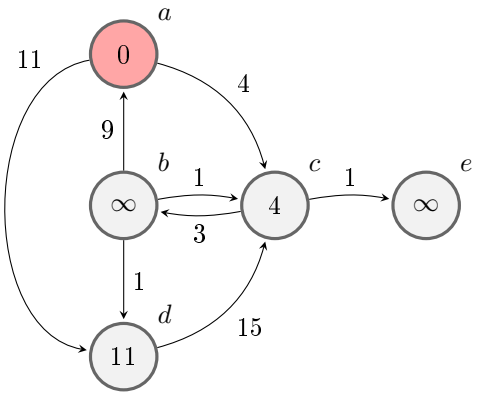


Figura 4.8: Si analizzano i nodi immediatamente raggiungibili da a e si assegna loro la distanza da a.

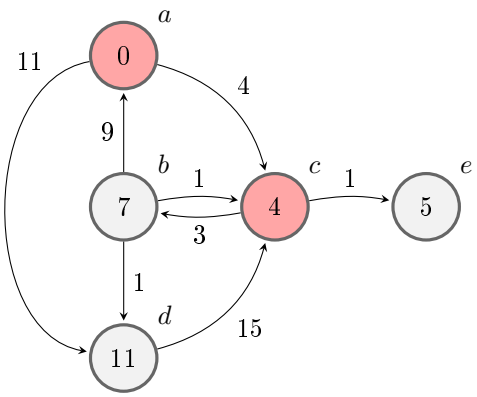


Figura 4.9: Si considera ora il nodo più vicino ad a, ossia il nodo c. Come fatto per il nodo a, si analizzano i nodi immediatamente raggiungibili da c e si assegna loro la distanza da c sommata alla distanza di c da a.

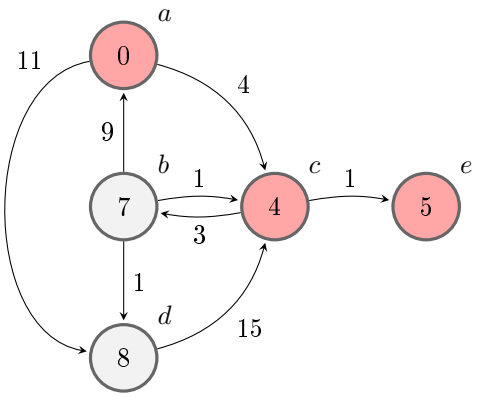


Figura 4.10: Si procede in modo analogo al passo precedente con il nodo e, poiché è quello a distanza minore da a, escludendo i nodi precedentemente analizzati (ossia a e c). Questo nodo non porta a nessun altro nodo, dunque passiamo a considerare b.

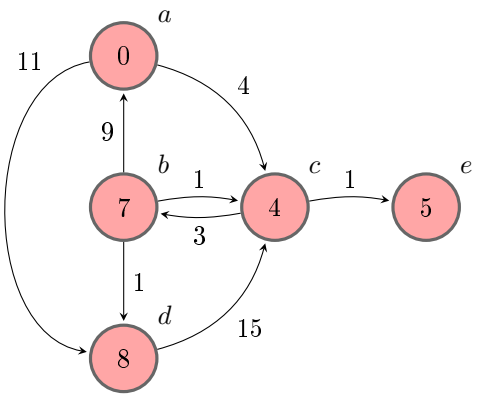


Figura 4.11: Da b si può tornare in c, ma non si migliora la distanza, e in d migliorando la distanza da 11 ad 8. Infine si procede analogamente con d, ma poichè da esso si può raggiungere solamente c e facendo ciò non si migliora la distanza dalla sorgente al nodo c, l'algoritmo termina perchè non ci sono più nodi collegati da considerare.

Per determinare il livello di fitness, come detto precedentemente con un valore tra 0 e 1, è necessario studiare non solo l'allineamento ottimale, ma anche quello peggiore. Per fare ciò ci basta considerare il caso in cui le transizioni del modello e del log sono completamente disallineate ad esempio:

$$\gamma_{1_w} = \begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline a & d & c & e & h & \gg & \gg & \gg & \gg & \gg \\ \hline \gg & \gg & \gg & \gg & \gg & a & b & d & e & h \\ \hline \end{array}$$

Questo allineamento ha "mosse solo del log" e "mosse solo del modello". Volendo calcolare la fitness di σ_1 sulla rete N_1 facciamo per semplicità l'ipotesi che la funzione δ assegni costo 1 a tutte le mosse che costituiscono un disallineamento. La fitness è definita come segue:

$$\mathit{fitness}(\sigma, N) = 1 - \frac{\delta(\lambda^N_{opt}(\sigma))}{\delta(\lambda^N_{worst}(\sigma))}$$

Nel nostro esempio $\delta(\lambda^N_{opt}(\sigma_1)) = 2$, $\delta(\lambda^N_{worst}(\sigma_1)) = 10$ e quindi $\mathit{fitness}(\sigma_1, N_1) = 1 - \frac{2}{10} = 0.8$. Come nel paragrafo precedente possiamo estendere la nozione di fitness ad un log di eventi:

$$fitness(L, N) = 1 - \frac{\sum_{\sigma \in L} L(\sigma) \times \delta(\lambda^N_{opt}(\sigma))}{\sum_{\sigma \in L} L(\sigma) \times \delta(\lambda^N_{worst}(\sigma))}$$

Se consideriamo un log di eventi, come nel capitolo precedente, composto da 12 tracce σ_0 e 8 tracce σ_1 riusciamo a calcolarne la fitness secondo quest'altra definizione che si basa sugli allineamenti:

$$fitness(L, N_1) = 1 - \frac{12 \times 0 + 8 \times 2}{12 \times 10 + 8 \times 10} = 0.92$$

Notiamo che il valore ottenuto con questa definizione di fitness è diverso da 0,9333 ossia il valore ottenuto utilizzando il metodo del Token Replay.

4.4 Token Replay e Alignments a confronto

Esistono diverse differenze tra i due metodi di conformance checking presentati:

- Le informazioni fornite dal metodo degli allineamenti su come modificare il modello per avere una migliore fitness sono maggiori per quantità e grado di accuratezza.
- L'Alignments è maggiormente configurabile. Ciò è dovuto alla funzione costo δ, la quale consente di stabilire sul modello quali attività pesino maggiormente e, di conseguenza, punire più severamente qualora ci sia un disallineamento.
- L'Alignments è indipendente dal modello di rappresentazione, ossia qualsiasi modello di processo che preveda la presenza di stato iniziale e finale può essere usato (alcuni esempi sono i modelli YAWL, BPMN e reti causali). Il Token Replay invece può essere eseguito solamente su una rete di Petri e per passare altri modelli è quasi sempre necessaria una conversione.
- Il Token Replay fornisce una diagnostica deterministica. Utilizzando il metodo dell'Alignments invece possono essere trovati dall'algoritmo diversi allineamenti ottimali per una stessa traccia. Ciò non si ripercuote sul valore di fitness, ma influenza la diagnostica.

Capitolo 5

Replay cumulativo

In questa sezione viene presentato un metodo di conformance checking ideato dall'autore di questa relazione. La spiegazione presenterà, in sezioni distinte, il metodo, le proprietà che ne derivano facendo un confronto teorico con il metodo Token Replay ed infine un confronto pratico sempre con il Token Replay discutendo nel mentre dell'implementazione. Si è scelto di utilizzare il Token replay come metodo di confronto poiché affronta il problema del conformance checking con una modalità affine.

5.1 Spiegazione del metodo

Sia Σ un sistema P/T con debiti descritto dalla matrice backward Pre e dalla matrice forward Post, che formano la rete WF del modello su cui vogliamo fare conformance checking, e dalla marcatura iniziale m_0. Sia quindi $|S| = k$ il numero di posti della rete che abbiamo preso in esame e C la matrice di incidenza. Sia inoltre $\sigma = \langle t_1, t_2, \dots t_n \rangle$ una traccia appartenente al log di eventi L che è oggetto del conformance checking.
Definiamo la funzione $h : \mathbb{Z}^{|S|} \to \mathbb{N}$ come

$$h(v) = \sum_{i \in \{1 \dots k\}} {v_i}^2 \qquad \text{con } v_i \text{ componente i-esima di } v \tag{5.1}$$

e la funzione $deb : \mathbb{Z} \to \mathbb{N}$ come

$$deb(v_i) = \begin{cases} |v_i| & \text{se } v_i < 0 \\ 0 & \text{altrimenti} \end{cases} \tag{5.2}$$

che è possibile estendere nel caso di input vettoriale come $deb : \mathbb{Z}^{|S|} \to \mathbb{N}^{|S|}$ nel seguente modo

$$deb(\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_k \end{bmatrix}) = \begin{bmatrix} deb(v_1) \\ deb(v_2) \\ \vdots \\ deb(v_k) \end{bmatrix} \tag{5.3}$$

Le marcature m_j con $1 \leq j \leq n$ prodotte durante la riproduzione sequenziale della traccia σ nel nostro sistema possono essere definite come segue, ricordando che con m_0 indichiamo la marcatura iniziale:

$$m_j = m_{j-1} + C[\bullet, t_j] \tag{5.4}$$

Consideriamo ora la successione di marcature $\{m_j\}$ generate riproducendo σ. Applichiamo ad ogni elemento di tale successione la funzione deb e poi la funzione h ottendo la successione di interi positivi $\{h(deb(m_j))\}$. Possiamo quindi calcolare la sommatoria dei valori presenti in quest'ultima successione:

$$\sum_{j=0}^{n} h(deb(m_j)) \tag{5.5}$$

Ciò che otteniamo è quindi la somma, per ogni marcatura raggiunta nella rete eseguendo la traccia σ, della somma dei quadrati dei debiti presenti in ogni posto. A questo punto ci domandiamo quanto può valere, nel caso peggiore, il valore discusso al punto precedente. Per fare ciò, anziché andare a considerare le marcature, andiamo a considerare il caso in cui tutte le transizioni appartenenti a σ, scattando, generino debiti in ciascuno dei pre-posti di ogni transizione. Definiamo quindi la successione $\{d_j\}$ che denota questo caso:

$$\{d_j\} = \begin{cases} d_0 = m_0 \\ d_j = d_{j-1} - Pre[\bullet, t_j] \end{cases} \tag{5.6}$$

Calcolando la sommatoria come nell'equazione (5.5) utilizzando però $\{d_j\}$ anzichè $\{m_j\}$, otteniamo la misura che stavamo cercando:

$$\sum_{j=0}^{n} h(deb(d_j)) \tag{5.7}$$

Si può quindi dare una prima definizione di fitness di una traccia e una rete, che tiene conto solamente del fattore "debiti".

$$fitness_{deb}(\sigma, N) = 1 - \frac{\sum_{j=0}^{n} h(deb(m_j))}{\sum_{j=0}^{n} h(deb(d_j))} \tag{5.8}$$

Esempio 14. Riprendiamo la rete WF N_1 oggetto dei nostri precedenti esempi e la traccia $\sigma_1 = \langle a, d, c, e, h \rangle$ e calcoliamo la $fitness_{deb}(\sigma_1, N_1)$.

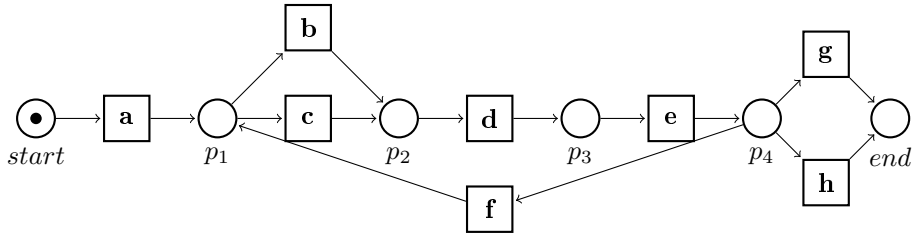


Figura 5.1: WF-net N_1 corrispondente al modello di processo.

Per fare ciò andiamo a considerare il vettore delle marcature col trascorrere delle transizioni. L'ordine dei posti all'interno del vettore è il seguente: $\begin{bmatrix} start & p_1 & p_2 & p_3 & p_4 & end \end{bmatrix}^T$. Avremo quindi:

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{\text{a}} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{\text{d}} \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{\text{c}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{\text{e}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \xrightarrow{\text{h}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

che rappresenta la successione $m_0 \to m_1 \to m_2 \to m_3 \to m_4 \to m_5$. Applichiamo ora la funzione *deb* ad ogni marcatura del passaggio precedente. Avremo:

$$\begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} \xrightarrow{a} \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} \xrightarrow{d} \begin{bmatrix}0\\0\\1\\0\\0\\0\end{bmatrix} \xrightarrow{c} \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} \xrightarrow{e} \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} \xrightarrow{h} \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix}$$

che rappresenta la successione $deb(m_0) \to deb(m_1) \to deb(m_2) \to deb(m_3) \to deb(m_4) \to deb(m_5)$. Ora calcoliamo tutti i valori di $\{h(deb(m_j))\}$. Avremo: $0 \to 0 \to 1 \to 0 \to 0 \to 0$. Possiamo ora calcolare agevolmente $\sum_{j=0}^{n} h(deb(m_j)) = 0+0+1+0+0+0 = 1$.

Ora facciamo con la successione $\{d_j\}$ la stessa cosa che abbiamo fatto con la successione delle marcature $\{m_j\}$.

$$\begin{bmatrix}1\\0\\0\\0\\0\\0\end{bmatrix} \xrightarrow{a} \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} \xrightarrow{d} \begin{bmatrix}0\\0\\-1\\0\\0\\0\end{bmatrix} \xrightarrow{c} \begin{bmatrix}0\\-1\\-1\\0\\0\\0\end{bmatrix} \xrightarrow{e} \begin{bmatrix}0\\-1\\-1\\-1\\0\\0\end{bmatrix} \xrightarrow{h} \begin{bmatrix}0\\-1\\-1\\-1\\-1\\0\end{bmatrix}$$

che rappresenta la successione $d_0 \to d_1 \to d_2 \to d_3 \to d_4 \to d_5$. Applichiamo quindi la funzione deb a ciascun membro della successione del passaggio precedente. Avremo:

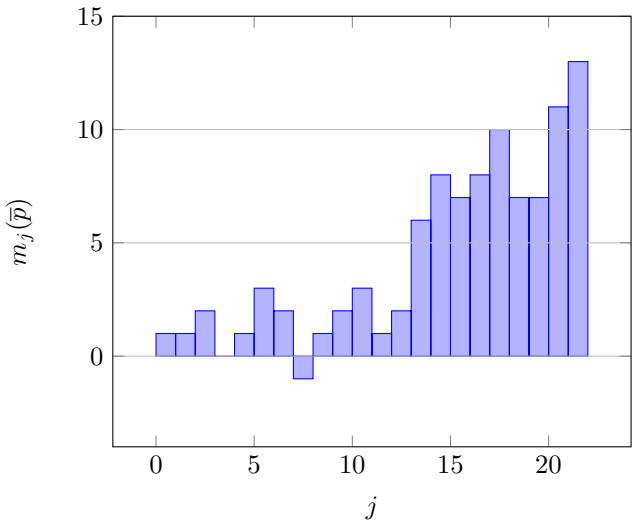
$$\begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} \xrightarrow{a} \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} \xrightarrow{d} \begin{bmatrix}0\\0\\1\\0\\0\\0\end{bmatrix} \xrightarrow{c} \begin{bmatrix}0\\1\\1\\0\\0\\0\end{bmatrix} \xrightarrow{e} \begin{bmatrix}0\\1\\1\\1\\0\\0\end{bmatrix} \xrightarrow{h} \begin{bmatrix}0\\1\\1\\1\\1\\0\end{bmatrix}$$

che rappresenta la successione $deb(d_0) \to deb(d_1) \to deb(d_2) \to deb(d_3) \to$

$deb(d_4) \rightarrow deb(d_5)$. Calcoliamo quindi tutti i valori di $\{h(deb(d_j))\}$. Avremo: $0 \rightarrow 0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4$. Possiamo ora calcolare agevolmente $\sum_{j=0}^{n} h(deb(d_j)) = 0{+}0{+}1{+}2{+}3{+}4 = 10$. Possiamo quindi finalmente calcolare la $fitness_{deb}(\sigma_1, N_1)$:

$$fitness_{deb}(\sigma_1, N_1) = 1 - \frac{\sum_{j=0}^{n} h(deb(m_j))}{\sum_{j=0}^{n} h(deb(d_j))} = 1 - \frac{1}{10} = 0.9$$

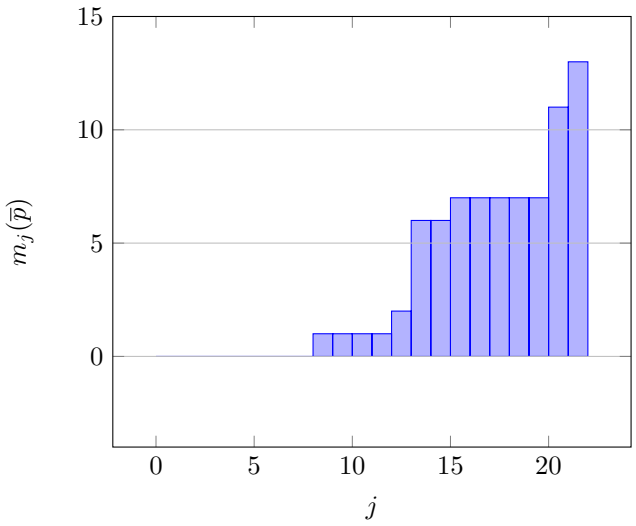
In seguito viene introdotta una definizione che tiene conto delle marche che non vengono mai consumate; successivamente queste due verranno unite insieme per dare una definizione totale di fitness di una traccia e una rete, ed infine della fitness tra un log di eventi ed una rete. Prendiamo in esame un certo posto $\overline{p}$. Disegniamo il grafico che mostra il numero di marche presenti in $\overline{p}$, ad ogni transizione di una traccia $\sigma = \langle t_1, t_2, \ldots t_n \rangle$ (t_0 si considera il momento iniziale in cui non è ancora scattata nessuna transizione).



Definiamo $z_j(\overline{p})$ come:

$$z_j(\overline{p}) = \begin{cases} m_j(\overline{p}) & \text{se } m_j(\overline{p}) \geq 0 \wedge m_j(\overline{p}) \leq m_x(\overline{p}), \text{ tale che } j < x \leq n \\ m_{j-1}(\overline{p}) & \text{altrimenti} \end{cases} \tag{5.9}$$

$z_j(\overline{p})$ mostra come si accumulano, al variare delle transizioni, le marche che non verranno mai consumate durante l'esecuzione della traccia. In seguito il grafico di $z_j(\overline{p})$



Notiamo che $z_j(\overline{p})$ è per costruzione monotona non decrescente e a valori positivi. Così come abbiamo definito la $z_j(\overline{p})$ per uno specifico posto $\overline{p}$, possiamo definirla per un vettore di posti, in particolare il vettore contenente i posti della nostra rete N su cui vogliamo fare conformance checking. Dunque:

$$z_j = \begin{bmatrix} z_j(p_1) \\ z_j(p_2) \\ \vdots \\ z_j(p_k) \end{bmatrix}$$

Bisogna però fare una precisazione riguardante il posto finale della rete nel momento in cui è avvenuta l'ultima transizione: in una situazione di completa compatibilità tra log e traccia in quella posizione avremmo un token. Per far sì che questo token venga considerato come "previsto" è necessario sottrarlo se presente (o se presenti più di uno) e quindi avremmo che $z_n = \begin{bmatrix} z_n(p_1) & z_n(p_2) & \dots & z_n(p_k) - 1 \end{bmatrix}^T$, specificando però che il posto p_k sia il posto di output della rete WF in esame, come descritto nella definizione 13 della sezione 2.3.

Possiamo dunque calcolare in modo analogo a come è stato fatto nei passi precedenti, ma escludendo la funzione *deb* poichè abbiamo già valori positivi e di interesse, la sommatoria:

$$\sum_{j=0}^{n} h(z_j) \tag{5.10}$$

Analogamente al discorso fatto per i debiti, dobbiamo domandarci quale sia il caso peggiore per la rete N, che massimizza il valore di $\sum_{j=0}^{n} h(z_j)$. La situazione che descrive questo caso è quella in cui ogni marca prodotta riproducendo σ su N è una marca che non verrà mai consumata. Abbiamo quindi:

$$\{r_j\} = \begin{cases} r_0 = m_0 \\ \\ r_j = r_{j-1} + Post[\bullet, t_j] \end{cases} \tag{5.11}$$

Calcolando

$$\sum_{j=0}^{n} h(r_j) \tag{5.12}$$

otteniamo la misura che stavamo cercando. Si può quindi dare la definizione di fitness di una traccia e una rete, che tiene conto solamente del fattore "marche non consumate".

$$fitness_{rem}(\sigma, N) = 1 - \frac{\sum_{j=0}^{n} h(z_j)}{\sum_{j=0}^{n} h(r_j)} \tag{5.13}$$

Finalmente possiamo definire la fitness complessiva di una traccia σ su N come:

$$fitness(\sigma, N) = \frac{fitness_{deb}(\sigma, N) + fitness_{rem}(\sigma, N)}{2} \tag{5.14}$$

e la fitness di un log di eventi L su N come:

$$fitness(L, N) = \frac{\sum fitness(\sigma_i, N) * freq_{ass}(\sigma_i)}{\sum freq_{ass}(\sigma_i)} \tag{5.15}$$

dove $freq_{ass}(\sigma)$ è il numero totale di volte che la traccia σ è presente nel log.

Esempio 15. Potendo ora continuare l'esempio precedente calcoliamo la $fitness_{rem}(\sigma_1, N_1)$. Per fare ciò andiamo a considerare la successione $\{z_j\}$. Avremo quindi:

$$\begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} \xrightarrow{a} \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} \xrightarrow{d} \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} \xrightarrow{c} \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} \xrightarrow{e} \begin{bmatrix}0\\0\\0\\0\\0\\0\end{bmatrix} \xrightarrow{h} \begin{bmatrix}0\\0\\0\\0\\0\\1-1\end{bmatrix}$$

che rappresenta la successione $z_0 \to z_1 \to z_2 \to z_3 \to z_4 \to z_5$. Ora calcoliamo tutti i valori di $\{h(z_j)\}$. Avremo: $0 \to 0 \to 0 \to 0 \to 0 \to 0$. Possiamo quindi calcolare agevolmente $\sum_{j=0}^{n} h(z_j) = 0+0+0+0+0+0 = 0$.

Successivamente facciamo con la successione $\{r_j\}$ la stessa cosa che abbiamo fatto con la successione $\{z_j\}$.

$$\begin{bmatrix}1\\0\\0\\0\\0\\0\end{bmatrix} \xrightarrow{a} \begin{bmatrix}1\\1\\0\\0\\0\\0\end{bmatrix} \xrightarrow{d} \begin{bmatrix}1\\1\\0\\1\\0\\0\end{bmatrix} \xrightarrow{c} \begin{bmatrix}1\\1\\1\\1\\0\\0\end{bmatrix} \xrightarrow{e} \begin{bmatrix}1\\1\\1\\1\\1\\0\end{bmatrix} \xrightarrow{h} \begin{bmatrix}1\\1\\1\\1\\1\\1\end{bmatrix}$$

che rappresenta la successione $r_0 \to r_1 \to r_2 \to r_3 \to r_4 \to r_5$.

Ora calcoliamo tutti i valori di $\{h(r_j)\}$. Avremo: $1 \to 2 \to 3 \to 4 \to 5 \to 6$. Possiamo a questo punto calcolare agevolmente $\sum_{j=0}^{n} h(r_j) = 1+2+3+4+5+6 = 21$. Possiamo quindi finalmente calcolare la $fitness_{rem}(\sigma_1, N_1)$:

$$fitness_{rem}(\sigma_1, N_1) = 1 - \frac{\sum_{j=0}^{n} h(z_j)}{\sum_{j=0}^{n} h(r_j)} = 1 - \frac{0}{21} = 1$$

Saremmo potuti arrivare allo stesso risultato anche fermandoci al calcolo del numeratore o, prima ancora, notando che non rimaneva alcun token riproducendo la traccia sul modello, se non nel posto finale dopo l'ultima transizione. Risulta quindi complessivamente

$$fitness(\sigma_1, N_1) = \frac{fitness_{deb}(\sigma_1, N_1) + fitness_{rem}(\sigma_1, N_1)}{2} = \frac{0.9+1}{2} = 0.95$$

5.2 Confronto teorico con Token Replay

Il metodo presentato ha diverse analogie e con il metodo Token Replay, ma anche molte differenze.

- Il modello di rete su cui è effettuato il Token Replay classico è una rete P/T, mentre quello su cui è effettuato il Replay cumulativo è una rete P/T con debiti. Questa grande differenza viene però in parte colmata, nel Token Replay classico, col fatto di avere un contatore di marche mancanti che svolge una funzione paragonabile a quella dei debiti nelle rete P/T con debiti.

- Mentre nel Token Replay classico viene fatto un calcolo solamente sui valori totali di token mancanti, prodotti, rimanenti e consumati alla fine dell'esecuzione e quindi come questi valori si siano sviluppati col procedere delle transizioni non influisce sul valore di fitness, nel metodo Replay cumulativo si tiene conto dei valori passo per passo col procedere delle transizioni.

- Nel Token Replay classico, se vi è una marca mancante verso l'inizio della traccia, o alla fine, ciò è ininfluente sul valore di fitness(vedi punto precedente). Nel replay cumulativo invece, avere un debito all'inizio può andare ad inficiare sul livello di fitness in maniera più massiccia piuttosto che un debito creatosi verso la fine della traccia. Questa proprietà è per certi aspetti criticabile, ma è giustificata da un'altra interessante proprietà e comunque controllata grazie al quadrato dell'equazione (1). Spieghiamo più nel dettaglio i due punti presenti nell'ultima frase. La proprietà interessante è che una situazione di debito, influisce sul sistema non solo in quanto tale ma tiene conto anche di quanto questo debito permanga col passare delle transizioni. Immaginiamo per esempio la traccia $\langle t_1, t_2, \ldots, t_k, t_{k+1}, \ldots, t_h, \ldots, t_n \rangle$ in cui la transizione t_k crea un debito in un posto p, mentre la transizione t_h fa tornare il posto p ad un numero non negativo di marche: ebbene, quanto più t_h si trova vicino a t_k, più la fitness ne beneficia. Se però il debito non viene mai recuperato allora è

verificata la situazione in cui un debito all'inizio di una traccia influenza di più di un debito alla fine. L'equazione (1) invece, grazie al quadrato, permette di marginare il problema appena presentato. Poniamo ad esempio l'attenzione su un posto p e ipotizziamo che sia l'unico sul quale compaiano debiti; sia la successione $\{m_j\}$ delle marcature in quel posto così determinata $\{m_j\} = \langle 1, -2, -2, -2, -2, -2, -2\rangle$. Allora avremo a numeratore nel calcolo della $fitness_{deb}$: $0^2 + 2^2 + 2^2 + 2^2 + 2^2 + 2^2 + 2^2 = 24$. Consideriamo invece ora la successione $\{m_j\}$ delle marcature in p $\{m_j\} = \langle 1, 0, 1, 2, 2, 3, -5\rangle$. Allora avremo a numeratore nel calcolo della $fitness_{deb}$: $0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 5^2 = 25$, un valore molto simile a quello ottenuto in precedenza. Se proviamo a fare lo stesso esempio senza utilizzare i quadrati avremo nel primo caso un valore totale di 12 e quindi, per avere circa lo stesso numeratore la seconda sequenza sarebbe dovuta essere $\{m_j\} = \langle 1, 0, 1, 2, 2, 3, -12\rangle$, accrescendo così maggiormente la disparità tra i debiti(non "saldati") creati all'inizio e quelli creati alla fine della traccia. L'intero discorso discusso in questo punto può essere fatto in modo del tutto analogo per le marche che non vengono consumate.

- Il replay cumulativo, considerando il quadrato dei debiti in ogni posto, va a punire maggiormente situazioni in cui sono presenti tanti debiti in un singolo posto o in pochi posti, piuttosto che situazioni in cui, a parità di numero di debiti, si hanno debiti più sparsi tra i vari posti. Questo può andare a vantaggio della diagnostica che viene fatta sulla rete perché le situazioni di disparità tra log e rete, dovute alla presenza di alcuni posti "errati"(male collegati o che non sarebbero dovuti proprio esistere) nel modello, vengono risaltate dalla presenza del quadrato e quindi più facilmente individuabili. Sempre sulla stessa linea di pensiero: meglio una rete imprecisa di poco tra i vari posti piuttosto che una rete in cui è presente una grave imprecisione in pochi posti, mentre gli altri posti sono precisi. Il concetto di precisione *su un insieme di posti* è dettato da quanto questi posti creino situazioni di debito o situazioni in cui si creino marche che non verranno consumate.

- Entrambi i metodi sono dipendenti dal modello di rappresentazione: le reti di Petri.
- Entrambi i metodi sono deterministici e quindi le diagnostiche che ne derivano sono sempre le medesime.

5.3 Confronto pratico con Token Replay ed implementazione

Oltre ad un confronto teorico, si è scelto di vedere dal punto di vista pratico come i valori di fitness derivanti dai metodi "Token Replay" e "Replay cumulativo" si rapportassero tra di loro. Per fare ciò sono stati utilizzati dei tool di python della libreria PM4Py [6]. In particolare PM4PY è stato utilizzato per il process discovery della rete (processo che preso un log di eventi crea in modo automatico una rete di Petri che rappresenta il modello associato al log) e per il calcolo della fitness tramite il Token replay. Ciò che invece è stato sviluppato completamente è un parser per esportare da Python in Java i log ed i modelli e, il metodo Replay cumulativo. Per il parser si è proceduto nel seguente modo: tramite i tool della libreria PM4Py è stato possibile accedere alle strutture dati delle reti e dei log, e questo accesso è stato sfruttato per produrre dei file txt che fossero strutturati in un determinato modo e rappresentassero reti e log in un formato compatibile con la lettura da parte del programma Java che era stato predisposto. Quest'ultimo quindi andava a leggere i file txt creati precedentemente e ricreava le strutture dati in Java secondo la logica del programma. È stato scelto di procedere in questo modo, e non di sfruttare direttamente i tool PM4Py, per avere un maggiore controllo sulla gestione delle strutture dati dato che erano state implementate da zero e quindi il loro funzionamento era più chiaro e, al bisogno, poteva essere facilmente modificato. È stato necessario fare qualche piccola modifica all'implementazione del Replay cumulativo perché i tool di PM4Py non producevano una rete WF come modello, mentre il suddetto metodo era pensato per quel tipo particolare di rete. In particolare i denominatori all'interno delle espressioni $fitness_{rem}$ e $fitness_{deb}$ potevano risultare

nulli producendo un valore NaN, così si è scelto in tale casistica di considerare il denominatore uguale ad 1. Sfortunatamente anche il caso in cui veniva prodotta una rete(non WF) in cui alcune transizioni non erano collegate al resto della rete ha fatto in modo che il Replay cumulativo non gestisse al meglio la situazione, in quanto non progettato per questo tipo di rete. Per implementare le reti è stata presa a modello la notazione algebrica presentata nelle sezioni iniziali della relazione e quindi dal punto di vista pratico sono state utilizzate matrici e vettori di interi. I log di eventi che sono stati analizzati sono 5 "sintetici", ovvero creati appositamente per eseguire operazioni di process mining, e 5 provenienti dalla realtà. Questi log sono provenienti da [7] e [8]. Nella figura 5.2 è rappresentata una tabella riassuntiva dei risultati.

		Token Replay		Replay cumulativo
Event Log	Origine	average fitness	log fitness	fitness
Artificial Digital Photo Copier Event Log	sintetico	0.29400	0.26588	0.51626
Artificial structured control-flow event log	sintetico	0.62933	0.62789	0.64229
Data driven process discovery artificial event	sintetico	0.31616	0.30867	0.52725
Large bank transaction process	sintetico	0.98267	0.98265	0.99665
Loan application example	sintetico	1.00000	1.00000	1.00000
JUnit	reale	0.74829	0.74829	0.96882
Sepsis cases	reale	0.29400	0.26588	0.51626
Heuristic process discovery	reale	0.69872	0.56843	0.68361
BPIChallenge2013	reale	0.67223	0.64036	0.50023
Receipt phase of an environmental permit application process	reale	0.49371	0.47828	0.86160

Figura 5.2: Tabella riassuntiva dei risultati

Nonostante nella maggior parte dei casi analizzati la fitness media delle trac-

ce e la fitness del log (entrambe calcolate con il Token Replay) siano molto simili, ciò non è scontato che accada. Probabilmente è più appropriato paragonare la fitness del Replay cumulativo con la fitness media del Token Replay anziché che con la fitness del log perché i primi due sono più conformi tra loro come metodi; infatti, nel Replay cumulativo, come ultimo passaggio viene eseguita una media. Questi due valori confrontati producono uno scarto di ≈ 0.2 in 5 casi su 10 e di ≈ 0.37 in 1 caso su 10, mentre nei restanti casi si ottiene un valore pressoché uguale. Si suppone che tanto più le reti create non "assomiglino" a reti WF (un gran numero di transizioni scollegate o collegate solo parzialmente), tanto più, in genere, i valori di fitness ottenuti dal metodo Replay cumulativo si discostino da quelli ottenuti dal Token Replay. Ciò è spiegabile poiché nel Replay cumulativo, come accennato, questa casistica è gestita solo passivamente e non è stata sviluppata ad hoc, producendo risultati meno precisi e quindi genericamente più distanti dai valori ottenuti con il Token Replay. A sostegno della suddetta ipotesi vi è anche il seguente fatto: sia P_1 il predicato "Il modello non assomiglia ad una rete WF" e sia P_2 "Lo scarto tra i valori di fitness ottenuto applicando i due metodi è ≥ 0.2 ". Si ha che il valore di verità di P_1 e P_2 è il medesimo in 8 casi su 10. Questo risultato, seppur non significativo statisticamente in quanto il numero di casi analizzato è troppo esiguo, fa presupporre che ci sia una certa correlazione tra questi due eventi.

Capitolo 6

Earth Movers' Stochastic Conformance Checking

Questa intera sezione si basa su [9] e per la parte generale riguardante la distanza di Earth Movers è stato utilizzato anche [10].

6.1 Introduzione e differenze di un approccio stocastico

In questa sezione verrà trattato un approccio al conformance checking radicalmente differente da quello delle sezioni precedenti poichè verrà introdotto il concetto di probabilità. In particolare chiameremo i modelli di processo che definiscono una certa probabilità per le proprie tracce *modelli (di processo) stocastici*. Il motivo per cui viene naturale rifarsi al concetto di probabilità in ambito di conformance checking apparirà evidente dal seguente esempio: immaginiamo di avere un log di eventi costituito da $[\langle a, b\rangle^1, \langle b, a\rangle^{99}]$, e un modello stocastico che riconosca le espressioni $[\langle a, b\rangle, \langle b, a\rangle]$ per cui in particolare viene associata alla traccia $\langle a, b\rangle$ una probabilità del 99% e alla traccia $\langle b, a\rangle$ una probabilità dell'1%. Per qualsiasi tecnica di conformance checking non stocastica si avrebbe una fitness ed una precisione perfetta tra log e modello, mentre, se si

considera la componente stocastica, essi hanno solamente il 2% in comune. Da ciò si può intendere che un approccio stocastico possa essere molto più accurato di quello standard ma, d'altra parte, ci sono diversi problemi a cui si può andare incontro, che verranno presentati a seguire. In questa sezione andremo quindi a presentare i fondamenti necessari al conformance checking stocastico e successivamente il metodo "Earth Movers' Stochastic Conformance Checking".

6.2 Concetti base

Definizione 18. *Sia Λ un alfabeto finito di attività(le componenti fondamentali di ogni traccia) e Λ^* l'insieme di tutte le possibili sequenze(le tracce) sull'alfabeto Λ. Un* linguaggio stocastico *è una collezione di tracce ognuna con la sua relativa probabilità. Formalmente un linguaggio stocastico è una funzione $f : \Lambda^* \to [0, 1]$ che associa ad ogni traccia una probabilità in modo tale che $\sum_{t \in \Sigma^*} f(t) = 1$.*

L'insieme $\tilde{M} = \{t \in \Lambda^* | M(t) > 0\}$ rappresenta l'insieme di tracce di un linguaggio stocastico M che hanno una probabilità non nulla.

Esempio 16. $[\langle a, b\rangle^{\frac{2}{3}}, \langle b, a\rangle^{\frac{1}{3}}]$ è un linguaggio stocastico composto da due tracce, le quali hanno rispettivamente $\frac{2}{3}$ e $\frac{1}{3}$ di probabilità di presentarsi.

Un log di eventi può essere facilmente ricondotto ad un linguaggio stocastico, semplicemente normalizzando la quantità delle tracce rispetto al numero totale delle tracce presenti nel log.

Esempio 17. Riprendendo ad esempio il log presente nell'introduzione segue che il linguaggio stocastico da esso definito risulta $[\langle a, b\rangle^{\frac{1}{100}}, \langle b, a\rangle^{\frac{99}{100}}]$, dove 100 è il numero totale di tracce del log.

La Earth Movers' Distance (o distanza di Wasserstein) è una misura della distanza tra due distribuzioni di probabilità su una regione D. Informalmente, immaginiamo di avere due cumuli di terra e di voler far assumere ad uno dei due la stessa distribuzione(forma) dell'altro. La Earth Movers' Distance indica il costo minimo per svolgere questo compito, in termini di quantità di terra spostata e distanza percorsa. Dal momento che noi stiamo confrontando due

distribuzioni di probabilità 1-dimensionali e discrete è necessario però pensare all'esempio presentato, adattandolo a questo caso più semplice.

Definizione 19. *Una rete di Petri stocastica etichettata generalizzata (GSLPN) è una tupla* $(P, T, F, \Lambda, l, T_i, T_t)$ *dove* (P, T, F, Λ, l) *è una rete di Petri etichettata,* $T_i \subseteq T$ *è un insieme di transizioni immediate e* $T_t \subseteq T$ *è un insieme di transizioni temporizzate tali per cui* $T_i \cap T_t = \emptyset$.

Le transizioni temporizzate non possono scattare se ci sono altre transizioni immediate abilitate. Ogni transizione immediata $t \in T_i$ ha un peso $w(t)$ e se ci sono più transizioni immediate abilitate, costituite dall'insieme $T' \subseteq T_i$, una transizione t può scattare con probabilità $w(t)/\sum_{t' \in T'} w(t')$. Una transizione temporizzata $t \in T_t$ ha una distribuzione esponenziale come tempo di abilitazione con parametro di sequenza di scatto $\lambda(t)$. Dato un insieme $T' \subseteq T_t$ la probabilità che una transizione $t \in T'$ scatti è $\lambda(t)/\sum_{t' \in T'} \lambda(t')$. Questo risultato è garantito dalla proprietà di assenza di memoria della distribuzione esponenziale. La probabilità di una traccia è la somma delle probabilità lungo tutti i percorsi sul modello che producono quella traccia. La probabilità di un singolo percorso che produce una traccia è calcolato moltiplicando tutte le probabilità incontrate lungo quel percorso. Possiamo quindi definire un linguaggio stocastico in base alla GSLPN che lo produce.

Esempio 18. Riportiamo l'esempio di [9].

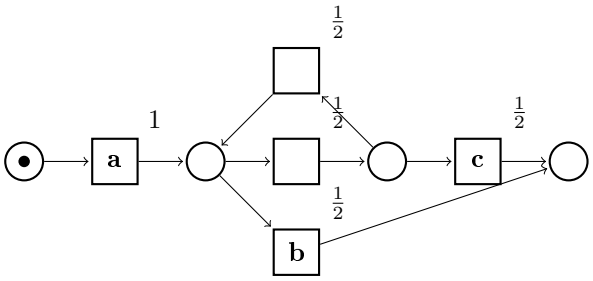


Figura 6.1: Esempio di GSLPN

Data la GSLPN in figura 6.1, contenente solo transizioni immediate, cerchiamo di risalire al linguaggio stocastico ad essa associato. Notiamo come prima

cosa che le uniche due tracce che possono essere riconosciute dal modello sono $\langle a, b\rangle$ e $\langle a, c\rangle$. Data la presenza di due transizioni silenti che formano un ciclo, abbiamo però a che fare con una possibilità di percorsi infinita. Andiamo a considerare la traccia $\langle a, b\rangle$: un percorso lo abbiamo direttamente facendo scattare le transizioni a e b avendo una probabilità associata a questo percorso pari a $1 \cdot \frac{1}{2}$, un altro percorso lo abbiamo includendo un ciclo delle due transizioni silenti, quindi $1 \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2}$ e così via includendo man mano sempre più cicli. Sommando insieme le probabilità ottenute abbiamo

$$\frac{1}{2} + \frac{1}{2}\frac{1}{2}\frac{1}{2} + \frac{1}{2}\frac{1}{2}\frac{1}{2}\frac{1}{2}\frac{1}{2} + \cdots = \sum_{n=0}^{\infty} \frac{1}{2}(\frac{1}{2})^n = \frac{\frac{1}{2}}{1 - \frac{1}{4}} = \frac{2}{3}$$

Per quanto riguarda la traccia $\langle a, c\rangle$ il discorso è simile e si ottiene quindi:

$$\frac{1}{2}\frac{1}{2} + \frac{1}{2}\frac{1}{2}\frac{1}{2}\frac{1}{2} + \frac{1}{2}\frac{1}{2}\frac{1}{2}\frac{1}{2}\frac{1}{2}\frac{1}{2} + \cdots = \sum_{n=0}^{\infty} \frac{1}{4}(\frac{1}{4})^n = \frac{\frac{1}{4}}{1 - \frac{1}{4}} = \frac{1}{3}$$

Otteniamo quindi il seguente linguaggio stocastico: $[\langle a, b\rangle^{\frac{2}{3}}, \langle a, c\rangle^{\frac{1}{3}}]$.

Si noti che non è banale riuscire a risalire al linguaggio stocastico e, nel caso ci siano transizioni silenti, non è stato ancora trovato un metodo che copra tutta la casistica possibile.

6.3 Spiegazione del metodo

In questa sezione verrà spiegato il metodo nel dettaglio, facendo riferimento all'idea intuitiva data precedentemente.

Definizione 20. *Siano L e M due linguaggi stocastici. La riallocazione è una funzione $r : \tilde{L} \times \tilde{M} \to [0, 1]$. $r(t, t')$ indica quanta massa di probabilità deve essere spostata da una traccia $t \in \tilde{L}$ ad una traccia $t' \in \tilde{M}$.*

La condizione per cui la somma di tutte le probabilità delle tracce deve dare come risultato 1 deve rimanere vera, pertanto:

$$\forall_{t \in \tilde{L}} L(t) = \sum_{t' \in \tilde{M}} r(t, t')$$

ed ugualmente la probabilità di ciascuna traccia $t' \in M$ deve essere mantenuta

$$\forall_{t' \in \tilde{M}} M(t') = \sum_{t \in \tilde{L}} r(t, t')$$

Definizione 21. *Sia la funzione* $d : \Lambda^* \times \Lambda^* \to [0, 1]$ *una funzione che definisca una distanza tra tracce: la coppia* (Λ^*, d) *costituisce uno spazio metrico [11], ossia una struttura matematica con le seguenti proprietà:*

$\forall x, y, z \in \Lambda^*$

- $d(x, y) \geq 0$
- $d(x, y) = d(y, x)$
- $d(x, y) \leq d(x, z) + d(z, y)$
- $d(x, y) = 0 \iff x = y$

Esempi di distanze possono essere:
- la distanza di Levenshtein, che indica il numero minimo di operazioni di inserimento, cancellazione o sostituzione necessario per trasformare una stringa in un'altra.
- la distanza unitaria, che restituisce un valore 0 se le due stringhe da confrontare sono uguali e il valore 1 se sono diverse.

Definizione 22. *Il costo per trasformare un linguaggio stocastico* L *in un linguaggio stocastico* M*, usando una funzione di riallocazione* r *è definito come il prodotto scalare tra la riallocazione e la distanza:*

$$cost(r, L, M) = r \cdot d = \sum_{t \in \tilde{L}} \sum_{t' \in \tilde{M}} r(t, t') d(t, t')$$

Possiamo dunque infine dare una definizione di Earth Movers' Stochastic Conformance Checking. Andando a prendere quella funzione di riallocazione r che minimizza il costo e sottraendo questo risultato ad 1 otteniamo una definizione di conformance checking, come un valore tra 0 (pessima corrispondenza) e 1(corrispondenza perfetta). Abbiamo quindi:

$$EMSC(L, M) = 1 - \min_{r \in R} cost(r, L, M)$$

dove R è l'insieme di tutte le funzioni di riallocazione tra il linguaggio L e il linguaggio M.

Uno dei principali problemi che si incontra quando si calcola la misura $EMSC$ è il fatto che si potrebbe avere a che fare con un linguaggio stocastico con un infinito numero di tracce. Per ridurre questo problema è stata introdotta una misura derivata dalla precedente: la truncated EMSC (tEMSC). Ciò che questa nuova misura cambia va a modificare è la condizione per la quale la somma di tutte le probabilità delle tracce appartenenti ad un linguaggio stocastico deve dare 1, infatti è necessario solamente che sia minore o uguale ad 1.

$$\forall_{t \in \tilde{L}} L(t) \leq \sum_{t' \in \tilde{M}} r(t, t')$$

Il senso di questo rilassamento è quello di accettare un valore anche minore di 1 come probabilità totale di un linguaggio stocastico, ma di poter rendere tale linguaggio finito, o, in altri casi, semplicemente più piccolo, con un chiaro vantaggio in termini di calcolabilità. Denotando con $m \leq 1$ il valore minimo di probabilità totale delle tracce che a priori ci si può porre e con M_m il linguaggio troncato derivato dall'esclusione di alcune tracce da M per far sì che la probabilità totale sia inferiore a 1 ma maggiore di m, si hanno le seguenti condizioni: $\Sigma_{t \in M_m} M_m(t) \geq m$ e $\forall_{t \in M_m} M_m(t) \leq M(t)$. Definiamo quindi:

$$tEMSC(L, M, m) = 1 - \min_{r \in R'} cost(r, L, M_m)$$

dove R' è l'insieme di tutte le funzioni di riallocazione tra il linguaggio L e il linguaggio M_m.

Si ottiene dunque il seguente risultato: siano L e M due linguaggi stocastici, allora con m che si avvicina ad 1 asintoticamente, $tEMSC$ e $EMSC$ coincidono: $EMSC(L, M) = \lim_{m \to 1} tEMSC(L, M, m)$.

Esempio 19. Presentiamo un esempio di Earth Movers' Stochastic Conformance Checking, come riportato in [9]. Consideriamo i linguaggi stocastici $L_e = [\langle a \rangle^{\frac{1}{4}}, \langle a, a \rangle^{\frac{3}{4}}]$ e $M_e = [\langle a \rangle^{\frac{1}{2}}, \langle a, a \rangle^{\frac{1}{4}}, \langle a, a, a \rangle^{\frac{1}{8}}, \langle a, a, a, a \rangle^{\frac{1}{16}}, \ldots]$ e una funzione di riallocazione r_e così definita:

r_e	$\langle a \rangle$	$\langle a, a \rangle$	$\langle a, a, a \rangle$	$\langle a, a, a, a \rangle$	$\langle a, a, a, a, a \rangle$	...
$\langle a \rangle$	$\frac{1}{4}$	0	0	0	0	...
$\langle a, a \rangle$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	...

Prendiamo ora come distanza d_l la distanza di Levenshtein normalizzata (distanza di Levenshtein diviso la lunghezza massima tra le tracce t e t'). Avremo quindi:

r_e	$\langle a \rangle$	$\langle a, a \rangle$	$\langle a, a, a \rangle$	$\langle a, a, a, a \rangle$	$\langle a, a, a, a, a \rangle$	...
$\langle a \rangle$	0	$\frac{1}{2}$	$\frac{2}{3}$	$\frac{3}{4}$	$\frac{4}{5}$	...
$\langle a, a \rangle$	$\frac{1}{2}$	0	$\frac{1}{3}$	$\frac{2}{4}$	$\frac{3}{5}$	...

Calcoliamo ora la funzione costo considerando la funzione di riallocazione r_e:

$$cost(r_e, L_e, M_e) = \frac{1}{4} \cdot 0 + 0 \cdot \frac{1}{2} + 0 \cdot \frac{2}{3} + 0 \cdot \frac{3}{4} + 0 \cdot \frac{4}{5} + \ldots$$

$$\frac{1}{4} \cdot \frac{1}{2} + \frac{1}{4} \cdot 0 + \frac{1}{8} \cdot \frac{1}{3} + \frac{1}{16} \cdot \frac{2}{4} + \frac{1}{32} \cdot \frac{3}{5} + \ldots$$

$$= \frac{1}{8} + \sum_{n=3}^{\infty} \frac{n-2}{2^n \cdot n} = \frac{13}{8} - \log 4 \approx 0.238706$$

Poiché r_e risulta essere non una riallocazione casuale, ma quella che minimizza il valore costo abbiamo che

$$EMSC(L_e, M_e) = 1 - cost(r_e, L_e, M_e) \approx 0.761294$$

Capitolo 7

Conclusioni e sviluppi futuri

Nel lavoro svolto sono stati affrontati i temi principali del process mining, ed in particolare del conformance checking. In una prima fase ci si è concentrati sulla presentazione dei modelli matematici necessari per comprendere le tecniche di conformance checking introdotte successivamente. In particolare sono state presentate le reti di Petri per analizzare le tecniche di conformance checking presenti in letteratura, e ho definito delle nuove reti di Petri chiamate reti di Petri con debiti che hanno costituito il modello matematico per un mio metodo di conformance checking. Sono stati dunque introdotti i due metodi del Token Replay e dell'Alignments e brevemente confrontati. Dopo la presentazione di tali tecniche si è proceduto con la spiegazione del metodo che ho sviluppato personalmente prendendo spunto dal Token Replay. Ho avuto anche modo di implementare il mio metodo e di sfruttare tool già esistenti e ciò ha richiesto un lavoro di coordinazione tra ciò che già esisteva e ciò che invece ho sviluppato. Sono stati poi messi in relazione dal punto di vista pratico, ma anche teorico, i risultati ottenuti con il mio metodo e il Token Replay. I due metodi hanno portato a risultati abbastanza simili, tranne nel caso in cui la rete prodotta utilizzando i tool di process discovery già esistenti risultava molto irregolare e strutturalmente molto differente rispetto ad una rete WF, particolare rete di Petri per cui il mio metodo era stato pensato. Ho scelto di sviluppare il mio metodo prendendo spunto dal Token Replay e non dall'Aligments perché

quest'ultimo metodo, nonostante abbia diversi pregi come il fatto di essere indipendente dal tipo di modello matematico con cui si rappresentano i processi, era poco compatibile con le possibilità introdotte dalle reti di Petri con debiti. Un'altra difficoltà a cui si sarebbe andati incontro implementando un metodo basato sull'Alignments è la necessità di risorse hardware: tale metodo infatti è impegnativo sia dal punto di vista computazionale che dal punto di vista del quantitativo di memoria necessaria.

Riguardo ai possibili sviluppi futuri del lavoro di stage, o in generale di aspetti in stretta relazione con quelli trattati e che meriterebbero un approfondimento, ci sono diversi temi di cui parlare. Innanzitutto potrebbe essere importante rivedere come dal punto di vista pratico tutti i temi affrontati possano essere applicati in ambito aziendale e in generale nel mondo del lavoro. Inoltre sarebbe interessante approfondire altri temi del process mining toccati solo in parte durante lo stage come ad esempio quello del process discovery. Per quanto invece riguarda un possibile ampliamento o miglioramento del lavoro già svolto, vi è la possibilità di introdurre nel metodo che ho sviluppato la gestione di transizioni silenti: nel mio metodo infatti si presupponeva che la rete in esame avesse come transizioni tutte e sole quelle che comparivano nel log, mentre alcuni algoritmi di process discovery generano reti dove, oltre alle transizioni normali, sono presenti alcune transizioni aggiuntive che non generano tracce nel log. La gestione di questa nuova casistica potrebbe essere quindi un arricchimento per il metodo sviluppato. Oltre a questa aggiunta invece, l'implementazione può essere resa più mantenibile, ricorrendo a tutte le strategie implementative derivanti dalla branca dell'ingegneria del software, ma anche più efficiente, nonostante in fase di sviluppo abbia già dedicato diverse attenzioni a questo aspetto.

Bibliografia

[1] T. Murata, "Petri nets: Properties, analysis and applications," tech. rep., PROCEEDINGS OF THE IEEE, VOL. 77, NO. 4, APRIL 1989.

[2] C. Girault and V. Rüdiger, *Petri nets for systems engineering : a guide to modeling, verification, and applications*. Springer-Verlag, 2001.

[3] W. van der Aalst, *Process Mining: Data Science in Action, 2nd edition*. Springer eBooks, 2016.

[4] J. Carmona, B. van Dongen, A. Solti, and M. Weidlich, *Conformance Checking: Relating Processes and Models*. Springer eBooks, 2018.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, 3rd edition*. McGraw-Hill, 2009.

[6] "pm4py." `https://pm4py.fit.fraunhofer.de/documentation\#conformance`.

[7] "processmining.org." `http://www.processmining.org/logs/start`.

[8] "tf-pm.ogm." `https://www.tf-pm.org/resources/xes-standard/about-xes/event-logs`.

[9] S. J. Leemans, A. F. Syring, and W. M. P. van der Aalst, "Earth movers' stochastic conformance checking," tech. rep., Queensland University of Technology, Brisbane, Australia and Process and Data Science (Informatik9) and RWTH Aachen University, D-52056 Aachen, Germany, 2019.

[10] "Earth mover's distance." http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/RUBNER/emd.htm.

[11] E. Sernesi, *Geometria 2*. Bollati Boringhieri, 2019.