Alma Mater Studiorum • Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea Triennale in Informatica

Implementazione di un Tool per l'Elaborazione di Unstructured Global Graph

Relatore:
Ivan Lanese

Presentata da:
Davide Schiavi

anno accademico 2017 - 2018

sessione III

Abstract

Global Graph è un linguaggio grafico utilizzato per modellare sistemi concorrenti. Il contributo fornito da questa tesi è rappresentato dallo sviluppo di Domitilla, un tool che permette l'elaborazione dei Global Graph. Le operazioni che Domitilla consente di fare sono la composizione parallela di due Global Graph e soprattutto la fusione di nodi all'interno dei Global Graph, che corrisponde a introdurre comunicazioni fra i partecipanti. Per effettuare la fusione è necessario generalizzare gli Structured Global Graph ottenendo degli Unstructured Global Graph. Domitilla consente inoltre di tradurre i Global Graph in reti di Petri.

Indice

1 Introduzione

1.1 Contesto

I sistemi comunicanti distribuiti sono una tipologia di sistema informatico, o di elaborazione dati, costituito da un insieme di processi interconnessi tra loro, le cui comunicazioni avvengono tramite uno scambio di messaggi opportuni. Tra le molteplici caratteristiche di un sistema comunicante, quella a cui facciamo riferimento è la concorrenza. La concorrenza rappresenta la possibilità di eseguire due o più istruzioni su macchine differenti contemporaneamente.

La gestione della concorrenza è però estremamente difficile perché non è possibile prevedere o determinare anticipatamente le decisioni che verranno prese all'interno di un sistema. Proprio per questo motivo non esistono strumenti che permettano di gestire in maniera semplice la sincronizzazione dei sistemi. Spesso accade, infatti, che all'interno di un sistema siano in esecuzione nello stesso istante più processi differenti. All'interno di un ipotetico modello, progettato alla perfezione, ci saranno moltissimi processi differenti che interagiranno tra loro e tutti quanti andranno a termine senza arrecare problemi o errori. Però, in realtà, l'esecuzione contemporanea di più processi conduce a delle interazioni e queste interazioni non sempre sono portate a termine, a volte, non consentono ad un processo di proseguire il proprio percorso prestabilito ma causano un blocco in attesa di qualcosa o di qualcuno che forse non arriverà mai. Queste attese, possono condurre a gravi problemi, tra i quali starvation, quando si hanno lunghe attese per l'accesso alle risorse, o deadlock, quando si hanno processi in attesa di risorse inesistenti. Eseguendo determinati e periodici controlli e gestendo in modo appropriato le risorse, questi problemi vengono evitati o tenuti sotto controllo. Queste tecniche sono però complesse e costose.

Nel nostro caso, però, vogliamo prevedere questi problemi studiando i sistemi comunicanti ad alto livello, prima ancora che siano realmente implementati. Serve dunque un modello formale che ci permetta di operare ad alto livello. Questo modello è rappresentato dai Global Graph [1] che sono proprio un linguaggio di alto livello, basato su grafi, che descrive le interazioni che avvengono tra i vari partecipanti di un sistema distribuito. Ottenuto questo modello, serve però un tool che consenta l'elaborazione dei Global Graph, questo tool è Domitilla. Domitilla ci consentirà di eseguire alcune operazioni sui Global Graph, queste operazioni sono la rappresentazione grafica, la composizione, la fusione, e la traduzione in reti di Petri.

1.2 Teoria alla base di Domitilla

Dato un Global Graph è possibile ottenere, attraverso l'operazione di proiezione, la descrizione di un partecipante la cui composizione con altri andrà a comporre un sistema comunicante distribuito. Lavorando ad alto livello grazie ai Global Graph è possibile eseguire su di essi alcune operazioni che ci permettono di mettere in concorrenza due o più grafi. Le operazioni a cui facciamo riferimento sono la composizione e la fusione. Attraverso la prima operazione, ovvero la composizione, è possibile simulare all'interno di un unico Global Graph due o più sistemi comunicanti che non possono però interagire tra loro; nel nostro caso specifico vedremo la composizione di solamente due grafi per volta. Quest'ultima, però, si limita semplicemente a mettere in parallelo tra loro i due sistemi. Attraverso la seconda operazione, ovvero la fusione dei nodi di un grafo, che è di fondamentale importanza per lo studio del loro comportamento, è possibile vedere come i due sistemi rappresentati potrebbero interagire tra loro. Questa operazione però conduce ad un grosso problema. Contrariamente alla composizione, dalla facile descrizione, la fusione di due Global Graph non è descrivibile come un Global Graph. Per ovviare a questo problema bisogna generalizzare i Global Graph descritti in letteratura. In particolare chiameremo Structured Global Graph i Global Graph descritti in letteratura e Unstructured Global Graph la generalizzazione che usiamo.

La fusione di due Unstructured Global Graph, infatti, produce un'Unstructured Global Graph che è descrivibile con la nuova sintassi adottata. Ottenuta la fusione di due o più Unstructured Global Graph sarà possibile considerare un ulteriore passaggio, ovvero la traduzione di un Global Graph in reti di Petri. Grazie a quest'ultima trasformazione sarà possibile ottenere ulteriori informazioni sui grafi.

1.3 Obiettivo

Nel corso di questa tesi vedremo come è stato sviluppata Domitilla, il tool di supporto, che ci permetterà di realizzare alcune operazioni sui Global Graph, tra le quali proprio la fusione. Siccome a causa della loro sintassi non è possibile descrivere la fusione degli Structured Global Graph, il tool implementa anche una funzione che permette la conversione da Structured Global Graph ad Unstructured Global Graph. Attraverso questa conversione saremo in grado di effettuare e rappresentare la fusione. Domitilla, il tool sviluppato, consente l'esecuzione di altre operazioni oltre alla fusione. Più precisamente, dato in input uno o due grafi è possibile ottenere come output:

- La semplice rappresentazione grafica;
- La conversione da Structured ad Unstructured;
- La composizione tra loro;
- La fusione dei nodi al loro interno;
- La traduzione in reti di Petri.

Tuttavia prima di procedere sarà necessaria una breve introduzione per comprendere come sia possibile lavorare sui Global Graph, utilizzati per la rappresentazione ad alto livello, dei sistemi comunicanti distribuiti.

2 Background

Vogliamo ora introdurre in breve, ma in modo preciso, alcuni dei concetti, delle definizioni e degli strumenti rivelatisi fondamentali e importanti per la realizzazione e per lo sviluppo di Domitilla, il tool di supporto. Questa breve introduzione sarà necessaria, inoltre, per la piena comprensione del documento.

2.1 DOT

DOT [4] è un linguaggio di descrizione per grafi, la cui estensione è *.gv*. Esso è il linguaggio che noi adotteremo per descrivere gli Unstructured Global Graph. DOT permette di ottenere una forte personalizzazione dei grafi che rappresenta; tuttavia, Domitilla, il tool sviluppato, si limita alla rappresentazione dei nodi, alla definizione della loro forma, e alla rappresentazione degli archi [5].

```
1  digraph parallel {
2          0 [label="" shape=circle]
3                  0 -> 1
4          1 [label="X -> Y : x" shape=rect]
5                  1 -> 2
6          2 [label="/" shape=square]
7                  2 -> 3
8                  2 -> 5
9          3 [label="A -> H : m" shape=rect]
10                 3 -> 4
11         4 [label="/" shape=square]
12                 4 -> 6
13         5 [label="D -> H : n" shape=rect]
14                 5 -> 4
15         6 [label="Y -> X : y" shape=rect]
16                 6 -> 7
17         7 [label="" shape=doublecircle]
18 }
```

Codice 1: Parallel Unstructured

Un esempio della sintassi è rappresentato sopra (cod.1). È possibile riconoscere ogni nodo (righe 2, 4, 6, ecc...), descritto da un numero che indica l'*id* e tra le parentesi quadre sono definite le sue caratteristiche di rappresentazione che ci limitiamo ad utilizzare; *label* per le etichette e *shape* per dare la forma al nodo. Sotto ad ogni descrizione sono indicati gli archi; nella forma, *id* nodo attuale in *id* nodo di destinazione ($id_{start} \rightarrow id_{end}$).

2.2 Global Graph

Global Graph è un linguaggio grafico e formale di specifica, utilizzato per modellare sistemi concorrenti a un alto livello di astrazione. Dalla proiezione di Global Graph su di un partecipante è possibile ottenere una descrizione del sistema comunicante distribuito.

Il modo più diretto e semplice di mostrare cos'è un Global Graph è illustrando com'è fatto; iniziando dalla rappresentazione grafica. La loro rappresentazione è simile al modello BPMN Choreographies [9] (Business Process Model and Notation). Le seguenti rappresentazioni mostrano in fig.1 un parallel e in fig.2 una choice.
Entrambi i grafi sono stati generati grazie al tool Domitilla.

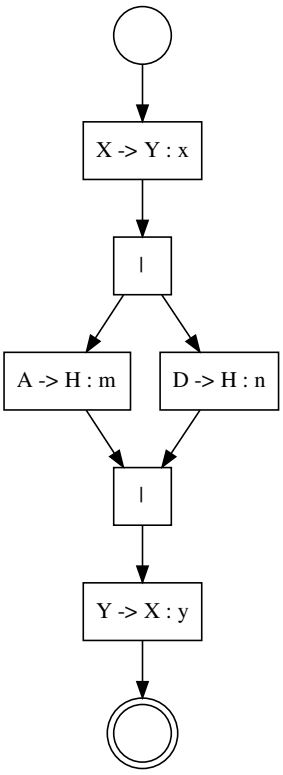

Figura 1: Parallel

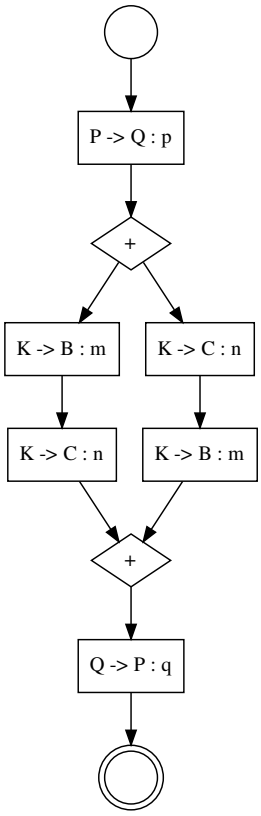

Figura 2: Choice

Osservando la fig.1, si possono distinguere i partecipanti che compongono il sistema, essi sono X, Y, A, B *e* H. Il sistema, innanzitutto, ha origine con un nodo sferico vuoto. Successivamente, si ha un'interazione che rappresenta X che invia ad Y il messaggio x, in seguito, il sistema si divide in due thread in parallelo (essi verranno eseguiti entrambi) in cui sono rappresentate altre interazioni contenenti altri messaggi. Terminato l'invio di entrambi i messaggi, i due thread si ricongiungono e viene inviato l'ultimo messaggio terminando poi con un nodo doppiamente cerchiato che indica la fine del sistema.

In fig.2 si vede un sistema dal comportamento simile, dove la principale differenza è rappresentata dai nodi di choice; essi rappresentano una scelta esclusiva.

I nodi del grafo vengono rappresentati in base alla loro funzione:

- I nodi etichettati con "start" vengono rappresentati con un nodo sferico vuoto;
- I nodi etichettati con un'interazione sono rettangolari e contengono l'interazione stessa. Nel nostro caso trattandosi di messaggi sono della forma $A \rightarrow B : msg$ e si legge: il partecipante A invia al partecipante B un messaggio msg.
- I nodi che rappresentano la scelta di uno dei flussi derivanti da uno split (choice) vengono rappresentati da un rombo contenente il simbolo "+";
- I nodi che rappresentano uno split concorrente del flusso (parallel) vengono rappresentati con un quadrato contenente il simbolo "|";
- Le iterazioni tra i nodi vengono rappresentati con archi orientati da un nodo a a un nodo b;
- I nodi etichettati con "end" vengono rappresentati con un nodo sferico doppiamente cerchiato vuoto.

Il nodo "start" non possiede alcun arco entrante, il nodo "end" non possiede alcun arco uscente conseguentemente il nodo "start" non può essere anche il nodo "end" e viceversa. Le istruzioni hanno esattamente un arco entrante e un arco uscente. I nodi di scelta possono avere più di un arco in entrata e più di un arco in uscita.

2.2.1 Structured Global Graph

Definiremo Structured tutti i Global Graph descrivibili con la seguente sintissi descritta. Di seguito vengono mostrati esempi di porzioni di codice che generano gli Structured Global Graph. Da notare che, in uno Structured Global Graph, due rami originati da un parallel si richiudono sempre su un parallel, lo stesso vale per le choice; questo è il motivo per cui non è possibile descrivere il prodotto della fusione.

```
1 X -> Y : x
2 |{
3         A -> H : m
4         |
5         D -> H : n
6 }
7 { Y -> X : y ; }
```

Codice 2: Parallel Structured

Il grafo mostrato in fig.1 è originato dal codice sovrastante (cod.2). Possiamo a prima vista notare le varie interazioni che indicano uno scambio di messaggi. Esse sono tutte quelle della forma $A \rightarrow B : msg$. La combinazione di caratteri "|{" in riga 2 indica l'inizio di un parallel. Il carattere "|" in riga 4 definisce la fine del ramo del parallel che è descritto fino alla riga precedente e l'inizio della descrizione del successivo ramo; in questo caso abbiamo due rami. I caratteri "{" e "}" rappresentano l'inizio e la fine di un set d'interazioni. Il carattere ";" è utilizzato per separare le interazioni nel caso in cui ce ne sia più di una inline oppure per definire la fine di una interazione.

```
1 P -> Q : p
2 +{
3         { K -> B : m ; K -> C : n ; }
4         +
5         { K -> C : n ; K -> B : m ; }
6 }
7 { Q -> P : q ; }
```

Codice 3: Choice Structured

Il grafo mostrato in fig.2 è originato dal codice sovrastante (cod.3). Si può notare che ci sono alcune piccole differenze la più importante e la combinazione di caratteri "+{" in riga 2, che rappresenta l'inizio di una choice. Il carattere "+" in riga 4 definisce la separazione tra i vari rami della choice; anche in questo caso ne abbiamo solamente due. Le altre differenze, ovvero la presenza dei caratteri "{", "}" che definiscono i set d'interazioni di ogni ramo e ";" che le separa tra loro.

2.2.2 Composizione

La composizione di due Global Graph permette di mettere in concorrenza due sistemi che si ha intenzione di descrivere e quindi poi di analizzarne i comportamenti. Generando una composizione è possibile rappresentare, all'interno di uno unico, due Global Graph che erano stati precedentemente progettati separatamente; essi vengono posti in parallelo. La composizione è ottenuta seguendo alcuni semplici passaggi; si realizzano un nodo iniziale e uno finale in comune ai due grafi, entrambi sono collegati, mediante un arco, rispettivamente a un nodo di apertura di un parallel, e a un nodo di chiusura di un parallel. Successivamente vengono modificati nodi d'inizio e di fine originali dei due grafi da comporre, le seguenti operazioni vengono quindi eseguite due volte, una sul primo grafo e una sul secondo grafo. Viene rimosso il nodo etichettato con "*start*" e al suo successore viene collegato uno dei due archi uscenti dal nodo di apertura del parallel precedentemente creata; similarmente viene ripetuta l'operazione sul nodo etichettato con "*end*", che viene rimosso e i suoi nodi predecessori vengono collegati con un arco al nodo di chiusura del parallel che conduce al nuovo nodo "end". Questa operazione però si limita semplicemente ad accostare i due grafi. Di seguito (fig.3) è riportato un esempio di composizione generato dai due grafi in fig.1 (riconoscibile sulla sinistra) e fig.2 (riconoscibile sulla destra). All'interno delle aree evidenziate è possibile vedere i nodi in comune di "start" e di "end", con i relativi parallel.

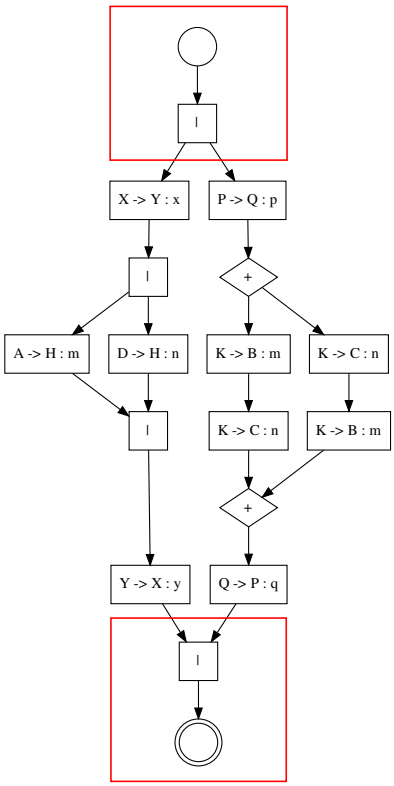

Figura 3: Prodotto della composizione di due grafi

2.2.3 Fusione

A differenza della composizione, la fusione dei nodi di un grafo, è di fondamentale importanza per l'analisi dei Global Graph. Come precedentemente detto attraverso i Global Graph è possibile definire dei modelli dai quali possiamo ottenere sistemi comunicanti distribuiti, i quali hanno la necessità scambiare messaggi tra loro, quindi sia di mandare che di ricevere messaggi. Se due Global Graph venissero messi in parallelo, molto probabilmente ci troveremo nella situazione in cui un nodo di un processo, a un determinato momento della sua esecuzione, abbia un partecipante A che necessita d'inviare un messaggio *m* a un altro partecipante X (fig.4); contemporaneamente all'interno dell'altro processo, un partecipante B si aspetta di ricevere un messaggio *m* da un partecipante Y (fig.5). Per semplicità le due interazioni in questione sono riportati di seguito.

Figura 4: Prima interazione

Figura 5: Seconda interazione

Questo è esattamente il caso ideale per poter effettuare la fusione delle due interazioni. Concettualmente, la fusione consente di eliminare le interazioni con i partecipanti *X* ed *Y*, perché a questo punto sono superflue, e di fare interagire direttamente i partecipanti *A* e *B*, che facendo riferimento all'esempio rappresentano i due estremi che si scambiano messaggi. Per poter eseguire la fusione è necessario soddisfare alcuni vincoli. Innanzitutto, entrambe le interazioni devono essere della stessa forma e nel nostro specifico caso che debbano rappresentare uno scambio di messaggi; devono quindi essere espresse come $A \rightarrow B : msg$. Inoltre, i due messaggi inviati devono essere uguali. Di seguito vediamo la fusione dei nodi in fig. 4 e in fig. 5.

Figura 6: Nodo fuso

Durante la creazione del nodo di fusione, vengono composte le informazioni ottenute dalla prima e dalla seconda interazione; il nuovo mittente è il medesimo della prima interazione fusa, il destinatario è il medesimo della seconda interazione fusa, il messaggio, essendo uguale per entrambi, non ha importanza da quale delle due interazioni è prelevato. Creato il nodo di fusione contenente la nuova interazione, bisogna effettuare le opportune modifiche ai grafi originali, per poterlo connettere nel modo corretto, infatti, non basta semplicemente sostituirlo ai vecchi nodi perché così facendo causeremmo confusione all'interno della descrizione del nostro Global Graph. I nodi source, da cui deriva il nuovo nodo creato, andranno certamente rimossi dal grafo, ma con criterio; saranno dunque sostituiti da una complessa rete di archi e nodi dove al centro è presente proprio il nodo di fusione. Questa complessa rete generata consentirà il corretto proseguimento dei partecipanti che si vogliono descrivere e di conseguenza del si-

stema comunicante. Questa nuova rete complessa di nodi e archi generata dalla fusione è detta *spiders*. Questa operazione è teoricamente eseguibile sugli Structured Global Graph, purtroppo però, la loro sintassi non permette di esprimere il concetto di spiders perché non è possibile descriverli definendo una struttura. Occorre perciò, trovare una soluzione a questo problema. La soluzione è rappresentata dagli Unstructured Global Graph, e lo strumento che ci permette di elaborarli è Domitilla, il tool appositamente sviluppato a sostegno della tesi. Il seguente grafo è ottenuto da una rappresentazione di un Unstructued Global Graph; i quali verranno introdotti di seguito.

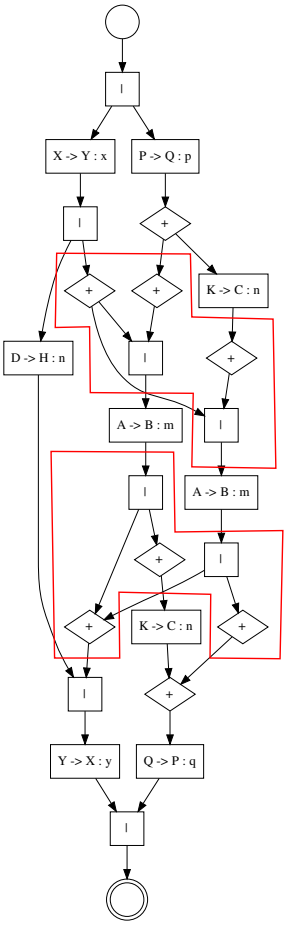

Figura 7: Prodotto della fusione di due grafi

La fig.7 mostra il prodotto della fusione applicata ai precedenti due grafi (fig.1 e fig.2) mediante il nodo $A \rightarrow H : m$ appartenente al primo grafo e ai nodi $K \rightarrow B : m$ appartenenti al secondo grafo. All'interno delle due aree rosse, si possono notare le due rappresentazioni grafiche degli spiders. Si può immediatamente notare come già da una fusione di pochi nodi, si generi una complessa rete di archi e nodi che non è rappresentabile attraverso la sintassi degli Structured Global Graph.

2.2.4 Unstructured Global Graph

Come detto precedentemente, non è possibile rappresentare una fusione utilizzando uno Structured Global Graph, per ovviare a questo problema è necessario introdurre il concetto di Unstructured Global Graph. Si ricorda che il precedente grafo (fig.7) è proprio un Unstructured Global Graph.

Domitilla, il tool che vedremo in seguito, riesce a definire le differenze tra i due metodi di descrizione grazie alla loro sintassi differente. La sintassi adottata è quella del formato DOT, che utilizziamo per gli Unstructured Global Graph. Di seguito riportiamo il codice che genera il grafo in fig.1 mostrato in precedenza, ma con la nuova sintassi. Essa generalizza quella utilizzata per descrivere gli Structured Global Graph. Infatti sia la porzione di codice in formato Structured Global Graph (cod.2) che la successiva in formato Unstructured global graph (cod.4) generano il medesimo risultato ovvero il grafo rappresentato in fig.1.

```
1  digraph choice {
2          0 [label="" shape=circle]
3                  0 -> 1
4          1 [label="P -> Q : p" shape=rect]
5                  1 -> 2
6          2 [label="+" shape=diamond]
7                  2 -> 4
8                  2 -> 5
9          3 [label="+" shape=diamond]
10                 3 -> 6
11         4 [label="K -> B : m" shape=rect]
12                 4 -> 8
13         5 [label="K -> C : n" shape=rect]
14                 5 -> 9
15         6 [label="Q -> P : q" shape=rect]
16                 6 -> 7
17         7 [label="" shape=doublecircle]
18         8 [label="K -> C : n" shape=rect]
19                 8 -> 3
20         9 [label="K -> B : m" shape=rect]
21                 9 -> 3}
```

Codice 4: Choice Unstructured

Nonostante i due formati conducano al medesimo risultato, la nuova sintassi, a differenza della precedente, ci permette di descrivere la complessa rete di nodi e archi generati dagli spiders.

2.3 Reti di petri

Le reti di Petri (Petri Nets)[11] sono uno dei più diffusi modelli di rappresentazioni matematiche di un sistema distribuito discreto. Permettono di rappresentare un sistema distribuito tramite un grafo bipartito e con annotazioni. Una rete di Petri PT (Place/Transition) è un grafo orientato caratterizzato da due tipi di nodi, *Place* (o posti) e *Transition* (o transizioni), connessi da archi diretti.

Graficamente le place sono rappresentate da cerchi e le transition da rettangoli. Gli archi possono collegare solamente due nodi di diverso tipo, non è possibile, quindi connettere tra loro due place o due transition. Una place avente un arco uscente indirizzato in una transition è detta di *input*, viceversa, un place avente un arco entrante da una transition è detta di *output*.

Ogni place può contenere una determinata quantità di *token*; le transition, quando ogni place di input relativa ad essa ha token disponibili, prende in input il token, lo consumano e lo restituiscno alle place di output. L'esecuzione delle reti di Petri è non deterministica.

Attraverso la marcatura di una rete di Petri possiamo determinare il grafo di raggiungibilità (reachability graph) e quindi trovare, se esiste, una sequenza che, partendo dalla place iniziale arriva alla place finale, o a qualsiasi altra place di interesse, tramite un qualunque percorso all'interno del grafo.

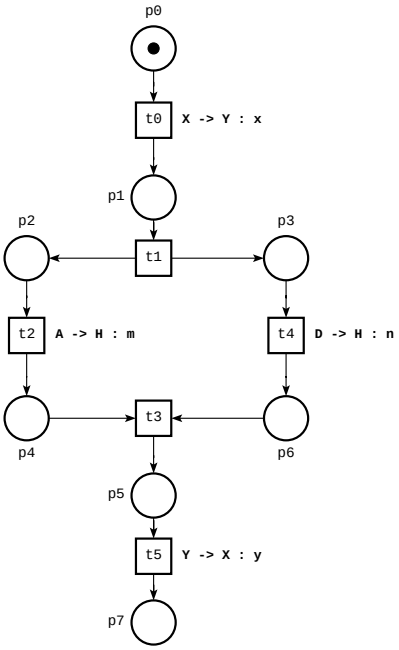

Figura 8: Parallel trdotto in rete di petri

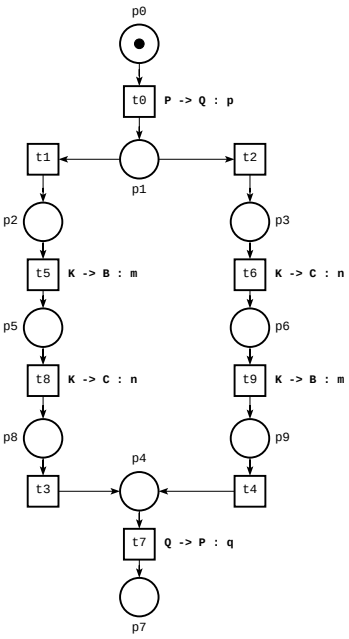

Figura 9: Choice trdotto in rete di petri

Nei precedenti grafi (fig.8 e fig.9) sono mostrate le rappresentazioni in reti di Petri dei grafi in fig.1 e fig.2. La traduzione è svolta dal nostro tool Domitilla, la rappresentazione da TINA [14]; descritta in seguito.

2.4 Traduzione nelle reti di Petri

Un'ulteriore operazione eseguibile sugli Unstructured Global Graph è la traduzione in rete di Petri. Questa caratteristica ci permette di eseguire analisi più approfondite sul grafo molto più velocemente e semplicemente. Le reti di Petri, infatti, sono un formato a lungo studiato e molto conosciuto, pertanto, a supporto di esso esistono differenti strumenti che consentono di fare studi, e non solo, su di essi. Eseguendo la conversione possiamo quindi usufruire una notevole quantità di software e strumenti già sviluppati e ampiamente testati per il raggiungimento del nostro scopo. In particolare, in questo progetto si farà uso del tool TINA per le rappresentazioni e le elaborazioni successive delle reti di Petri.

TINA (TIme petri Net Analyzer) è un tool studiato per la creazione, per la modifica e per l'analisi delle reti di Petri. Sviluppato inizialmente in OLC e poi successivamente da VerTICS, gruppo di ricerca di LAAS/CNRS. TINA ci è stato fondamentale per la rappresentazione grafica delle reti di Petri derivanti dai Global Graph e per alcune successive analisi, come ad esempio per la simulazione dei percorsi od ottenere grafi di raggiungibilità o altro.

Passando alla rappresentazione in reti di Petri, non si avrà nessuna perdita d'informazioni contenute nei Global Graph; al contrario manterremo intatte tutte le caratteristiche descritte. Si tratta quindi di una operazione eseguibile senza nessun effetto collaterale, anzi, a contrario ci predispone a un numero maggiore di analisi possibili. La traduzione è però dettata da rigide regole, le quali assicurano il corretto raggiungimento dell'obiettivo. Vediamo ora come sono definite le regole; nella seguente pagina è possibile vedere la rappresentazione grafica derivante dall'applicazione delle regole.

Le interazioni sono rappresentabili con una place di input avente un arco uscente indirizzato in una transition la quale a sua volta possiede un arco indirizzato in una place di output. L'area evidenziata in blu all'interno della fig.10 ne mostra il risultato.

Le choice e i parallel rappresentando delle scelte, sono caratterizzate da una rappresentazione in apertura e una in chiusura dove nel mezzo sono contenuti tutti i nodi che dipendono dalle scelte, quindi descriviamo come si comportano prima in apertura e poi in chiusura. I parallel in apertura, sono rappresentabili da una transition avente due archi uscenti in direzione di due palce di output differenti, le quali avranno un arco uscente ciascuna in direzione della rappresentazione del nodo successivo. La prima area evidenziata in verde all'interno della fig.10 ne mostra il risultato. In chiusura, sono rappresentabili con una transition avente due archi entranti dai nodi che precedono, ed un arco uscente indirizzato in una pace di output. La seconda area verde in arancione all'interno della fig.10 ne mostra il risultato.

Le choice in apertura, sono rappresentabili con una place di input avente due archi uscenti indirizzati in due transition differenti le quali hanno un arco uscente indirizzato alla rappresentazione del prossimo nodo. La prima area evidenziata in arancione all'interno della fig.10 ne mostra il risultato. In chiusura, sono rappresentabili con due transition differenti aventi un arco entrante dai nodi che le precedono, e un arco uscente indirizzato a una place di output comune. La seconda area evidenziata in arancione all'interno della fig.10 ne mostra il risultato.

La fig.10 è composta da due grafi, il primo a sinistra è un Global Graph rappresentato grazie a Domitilla; il secondo, a destra, è la traduzione del primo in rete di Petri, ottenuta grazie a Domitilla, rappresentata con TINA.

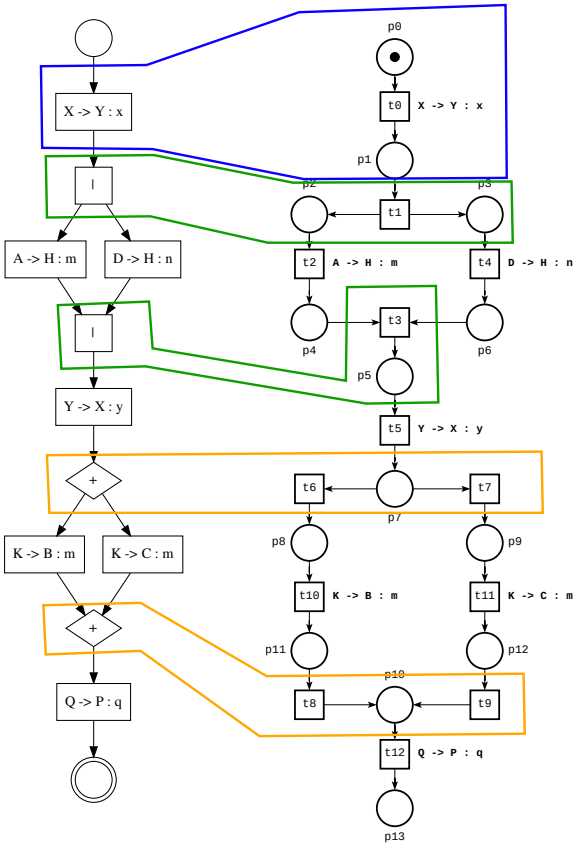

Figura 10: Esempio di conversione di un grafo

3 Domitilla, l'utilizzo

Domitilla, è il tool appositamente sviluppato per progredire nella realizzazione del progetto intrapreso con i Global Graph, ma, interrotto a causa dell'espressività limitata provocata dalla sintassi degli Structured Global Graph, che non consentono di descrivere in modo appropriato la complessa struttura di uno spider derivante da una fusione. Il principale scopo del nostro tool è quello di fornire uno strumento per automatizzare il processo di fusione tra i Global Graph; per poter arrivare a concludere questa operazione, Domitilla, si serve, però, di altre feature fondamentali. Indispensabile è, infatti, la conversione da Structured Global Graph ad Unstructured Global Graph, sui quali, poi, andranno eseguite tutte le funzioni richieste al tool, necessarie al proseguimento dello studio dei Global Graph.

Domitilla consente di eseguire alcune operazioni, esse sono: la rappresentazione grafica, la composizione, la fusione, e la conversione. Vediamo ora come si presenta e quali sono gli utilizzi possibili del tool sviluppato; all'interno della prossima sezione analizzeremo come lavora per procurare il risultato richiesto.

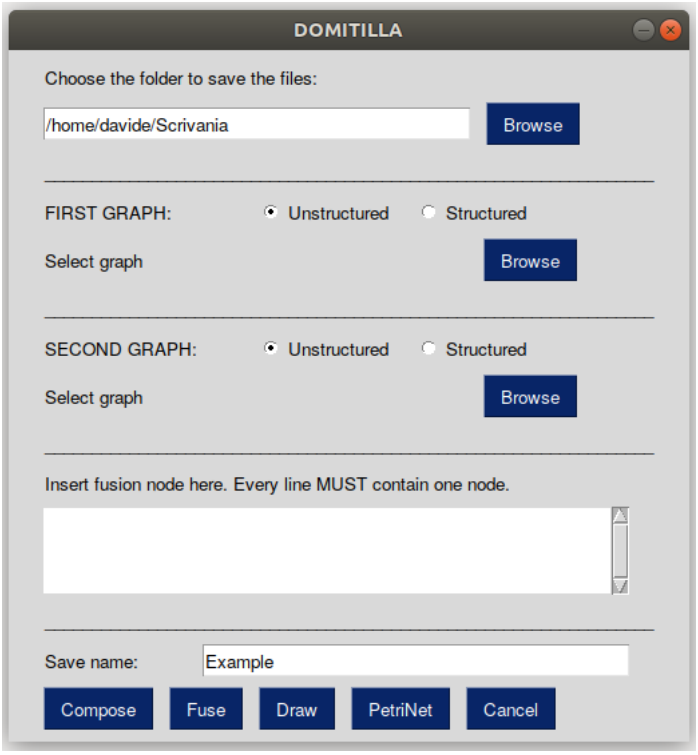

Figura 11: Screenshot del tool Domitilla

Per poter utilizzare il tool si richiede l'installazione di Python, di Graphviz e di PySimpleGUI. Python serve alla corretta compilazione del programma, Graphviz è la libreria che ci permette di disegnare i grafi e PySimpleGUI è la libreria che permette la realizzazione dell'interfaccia grafica.

3.1 Interfaccia

Il tool ha un aspetto semplice e minimale, dove si possono, fin da subito, notare cinque sezioni; la parte centrale è composta da tre sezioni dove è possibile interagire per inserire i dati necessari alla manipolazione dei grafi.

In alto è possibile modificare il path di salvataggio dei file, grazie ad una semplice ricerca della cartella di salvataggio.

Al centro ci sono tre sezioni, la prima è dedicata al "primo grafo", la seconda è dedicata al "secondo grafo", e la terza è dedicata all'inserimento dei nodi di fusione.

In basso è possibile cambiare il nome con cui salvare i file in output. Il nome inserito, tuttavia, verrà utilizzato solamente nel caso della composizione e della fusione. Immediatamente sotto sono presenti i bottoni che ci permettono di eseguire le funzioni richieste.

Da evidenziare il fatto che per poter usare nel modo corretto Domitilla occorre sempre utilizzare in input gli Unstructured Global Graph, tranne nel caso in cui si voglia ottenere solamente la rappresentazione grafica. L'unico modo per poter ottenere una conversione di un grafo in formato Structured, è attraverso la sua rappresentazione grafica, grazie alla quale, avremo in output, oltre al disegno, un file DOT contenente il medesimo grafo ma in formato Unstructured.

Tutti i grafi (fatta eccezione per le reti di Petri) rappresentati all'interno del documento sono ottenuti grazie a Domitilla.

3.2 Rappresentazione grafica

Il tool permette di selezionare fino a due file diversi, rispettivamente nelle sezioni "first graph" e "second graph"; dopo aver selezionato il file desiderato, mediante il bottone "Draw", otteniamo per ogni file precedentemente scelto, la rappresentazione grafica in formato PDF, in aggiunta, se il grafo di partenza era nel formato Structured Global Graph otteniamo in output un ulteriore file in formato DOT contenente il medesimo grafo ma convertito in formato Unstructured Global Graph. È quindi possibile utilizzare questa funzione per convertire un grafo da Structured Global Graph ad Unstructured Global Graph.

Nel caso in cui si volesse operare su un Global Graph nella formato Structured è necessario comunicarlo al tool attraverso la selezione adeguata del radio-button; esso, infatti, operando principalmente su Unstructured Global Graph, di default assume di operare tramite essi. Il processo di ottenimento della rappresentazione è infatti diverso tra i due formati. Nel caso in cui non venisse comunicato il corretto formato la rappresentazione fallirebbe e otterremmo un grafo vuoto.

3.3 Composizione

Selezionando i due file desiderati, nelle rispettive sezioni, e mediante il bottone "Compose", otterremo un file in formato DOT contenente l'Unstructured Global Graph relativo alla composizione dei due grafi e in formato PDF la rappresentazione grafica ottenuta dal precedente file.

Non è possibile ottenere direttamente la composizione utilizzando uno o due Structured Global Graph. È obbligatorio prima ricavare la loro conversione in Unstructured Global Graph, mediante la loro rappresentazione grafica; solo successivamente è possibile effettuare la composizione.

3.4 Fusione

Sono disponibili due modalità di utilizzo di questa funzione; la scelta, tra le due, dipende dalla numero di grafi in input selezionati dall'utente e viene eseguita automaticamente dal tool. Nel caso in cui venisse selezionato un solo grafo, permette di eseguire la fusione dei nodi contenuti al suo interno. Se invece venissero selezionati due grafi verrà eseguita prima la loro composizione, in modo tale da ottenere un unico grafo sul quale eseguire le operazioni, e solo successivamente, verrà eseguita sui nodi del grafo appena composto la fusione. Necessaria, oltre alla selezione dei grafi, è l'inserimento manuale, nello spazio appositamente dedicato, delle coppie dei label che sono all'interno nei nodi che verranno fusi. Se ipoteticamente non dovessero essere inseriti nodi da fondere, il programma non effettuerà nessuna modifica. Lo stesso verrà fatto nel caso in cui ci siano label mal scritti, oppure, nel caso in cui non ci sia alcun match tra i label indicati, perché, ad esempio i messaggi sono tutti differenti tra loro.

Dopo aver correttamente inserito i dati necessari, mediante il bottone "Fuse" otterremo un file in formato DOT contenente il nuovo Unstructured Global Graph con i nodi fusi e la relativa rappresentazione grafica in un file PDF.

3.5 Traduzione nelle reti di Petri

Domitilla, anche in questo caso, ci permette di selezionare fino a due grafi diversi, all'interno delle rispettive sezioni. Mediante la pressione del bottone "PetriNet" otterremo uno o due file, in base a quelli precedentemente scelti, in formato .ndr contenenti i medesimi grafi ma rappresentati le reti di Petri; .ndr è il formato usato da TINA. Il nuovo file ottenuto, essendo compatibile, è caricabile all'interno del tool TINA, grazie al quale potremo effettuare ulteriori operazioni.

4 Domitilla, l'implementazione

Avendo precedentemente esaminato tutte le modalità d'utilizzo del tool Domitilla dal punto di vista di un utente, andremo ora ad approfondire in particolare quali sono stati gli strumenti, le tecniche e gli algoritmi utilizzati per la realizzazione del tool, le scelte d'implementazione e come sono state scritte le varie funzioni.

Domitilla è stato sviluppato nel linguaggio Python. La scelta deriva dal fatto che Python mette a disposizione alcune comode librerie che ci permettono di generare la rappresentazione grafica. In particolare, la libreria utilizzata che ha condizionato la scelta del linguaggio di sviluppo è Graphviz. Il tool è stato sviluppato all'interno di PyCharm [12], un IDE appositamente realizzato per la programmazione in Python da JetBrains [8]. Domitilla conta all'incirca 950 linee di codice, suddivise tra i vari file che compongono il programma.

In aggiunta, essendo sviluppato in Python il tool dovrebbe essere eseguibile su tutti i sistemi operativi (Microsoft, Mac OSX, e Linux), tuttavia è stato solamente testato su Ubuntu 18.04.2 LTS.

Domitilla è un software libero con licenza MIT.
Disponibile al download all'indirizzo: https://github.com/dedo94/Domitilla.

4.1 Introduzione all'implementazione

Domitilla è principalmente basato sull'analisi e sull'interpretazione delle stringhe contenute nei file in input; essi vengono interpretati come file di testo e poi elaborati. Grazie a questa analisi è possibile ricavare ogni tipo di informazione necessaria per lavorare sui grafi. Per poter lavorare su diversi tipi di formati di grafi si è rivelato necessario elaborare un metodo per poterli prima rendere uguali a livello concettuale. A tal proposito si è ideata una struttura dati comune a tutti, dove poi si differenzia il metodo di riempimento, scelto in base al formato momentaneamente in uso. La struttura ideata consiste in un'array dove ad ogni indice è assegnato un nodo; ogni elemento dell'array contiene alcune informazioni tra cui i puntatori ai nodi successivi. Una volta riempita la suddetta struttura il tool lavora su di essa per realizzare le varie operazioni. Una volta terminate le operazioni Domitilla genera l'output desiderato ricavandolo dalla struttura.

La decodifica di uno Structured Global Graph è risultata più complessa perché più articolata; si tratta infatti di un linguaggio di specifica e come tale è necessario prima interpretarlo. Per poter comprendere il contenuto dei file è stato creato un parser, il quale ha come scopo quello di individuare le combinazioni di caratteri chiave che delimitano le varie istruzioni dei grafi, e decifrarle per poi rendere possibile il riempimento della struttura dati. Ottenuto lo strumento per interpretare il file non è stato difficile compilare la struttura dati nel modo corretto.

La decodifica di uno Unstructured Global Graph è risultata, invece, molto più semplice, perché per la loro descrizione è stato adottato il modello DOT, scelto appunto per la sua semplicità e per la sua sintassi standard facile da capire e da replicare. L'interpretazione di questo formato risulta quindi quasi immediata. Lavorando su questo formato, una volta interpretato il file, non è difficile compilare la struttura dati.

4.2 Struttura dati

È stata appositamente ideata una classe che permettesse di schematizzare nella stessa maniera tutti i grafi con cui si vuole lavorare. Esso ci permette di creare una struttura dati sulla quale è possibile interagire per tutta la durata del lavoro. In particolare si utilizza un'array di nodi con puntatori ai successivi; dove ogni elemento è nella forma descritta sotto (cod.5). I nodi contenuti nella struttura, non vengono generati immediatamente nella loro forma finale; essi vengono continuamente aggiornati con informazioni nuove finché l'analisi del file in input non termina. Nel nostro caso, ogni nodo facente parte del grafo è descritto dalla classe *node*, quindi, ogni volta che si trova una nuova informazione, o si scoprono modifiche da apportare, è sufficiente ricercare il nodo voluto all'interno della nostra struttura, anche in fase di completamento, e apportare le dovute modifiche. Tutte le operazioni eseguite da Domitilla sfruttano la struttura dati precedentemente citata; da questo ne deriva la vitale importanza della classe per il tool. Di seguito possiamo vedere come è scritta la struttura.

```
1  class node:
2      def __init__(self, id, istruction, *nxt_node):
3          self.id = id
4          self.ist = istruction
5          self.next_node = []
6          for i in nxt_node:
7              self.next_node.append(i)
8          if str(istruction).count("->") == 1
9                  and str(istruction).count(":") == 1:
10             strz = str(istruction).split("->")
11             self.snd = strz[0].strip()
12             strz = strz[1].split(":")
13             self.recv = strz[0].strip()
14             self.msg = strz[1].strip()
15         else:
16             self.snd = self.recv = self.msg = "null"
```

Codice 5: Struttura dei nodi

La struttura dati permette di salvare, all'interno dell'array, le seguenti informazioni: *id* che ci permette di identificare i nodi in modo univoco, *ist* che ci permette di identificare l'etichetta di un nodo e *next_node* un'array contenente i puntatori ai nodi immediatamente successivi. Gli *id* sono necessari perché le etichette dei nodi spesso si ripetono e non è possibile, perciò, identificare un nodo tramite l'etichetta contenuta. Se l'etichetta di un nodo rappresenta uno scambio di messaggi, è possibile ottenere direttamente il mittente (*snd*), il destinatario (*recv*) e il messaggio (*msg*).

4.3 Rappresentazione grafica

Dopo aver ottenuto la struttura dati, la rappresentazione è resa semplice dalla libreria Graphviz, è sufficiente derivarne il formato DOT che poi dato in input alla libreria, attraverso le funzioni *Digraph()* grazie alla quale è possibile inizializzare un grafo, *node()* grazie alla quale è possibile creare i nodi e *edge()* grazie alla quale è possibile creare gli archi, restituisce il grafo disegnato in PDF. Da notare che dal primo passaggio che viene eseguito, ovvero la derivazione nel formato DOT, otteniamo anche la traduzione del Global Graph dal formato Structured al formato Unstructured. Durante l'esecuzione di questa funzione non vengono apportate modifiche di alcun genere alla struttura, essa viene solamente letta.

La derivazione nel formato DOT la otteniamo leggendo la struttura dati. Per ogni struttura si genera un nuovo file (*.gv*) dove al suo interno si andranno a scrivere tutte le informazioni necessarie per ottenere il grafo in formato DOT. Si analizza la struttura, nodo per nodo, e si generano delle stringhe simili a quelle viste in precedenza (cod.4) basandosi sulle informazioni contenute. Successivamente vengono scritte all'interno del file. Una volta terminato il processo abbiamo ottenuto un Unstructured Global Graph.

4.4 Composizione

Essendo, la composizione, costituita da due grafi differenti, vengono generate due strutture dati, una relativa al primo grafo e una relativa al secondo grafo che vogliamo comporre. Durante la composizione si genera una nuova struttura dati dove al suo interno vengono inseriti quattro nuovi nodi, uno di "start", uno di "end" e due "parallel" rispettivamente uno di apertura e uno di chiusura. Prima di poter copiare le due strutture dati all'interno della nuova struttura creata, vengono ricalcolati tutti gli *id* dei due grafi e di conseguenza vengono ritoccati anche tutti i collegamenti rappresentati dagli archi. Per poter ricalcolare gli *id*, di grafi differenti, è stata scritta una funzione apposita, che permette di riassegnare in modo sequenziale e univoco gli *id*, all'interno della nuova struttura, che però derivano da grafi differenti. Gli *id* di ogni grafo partono, infatti, dal numero 0, quindi con la composizione si avrebbero dei problemi di sovrapposizione. Terminata questa operazione tutti i nodi dei due grafi dati in input fatta eccezione per i relativi nodi "start" ed "end", i quali vengono rimpiazzati dai nuovi precedentemente creati, vengono copiati all'interno della nuova struttura. Per finire il contenuto di quest'ultima rappresenta la composizione dei due grafi.

Durante l'esecuzione di questa funzione non vengono modificate direttamente le strutture dati originali ma se ne crea una terza la quale riporta quasi interamente le due date in input. Per generare e scrivere il file finale vengono, infatti, non vengono lette le strutture dati da cui derivano i due grafi originali ma si legge solamente la nuova. Derivandone da essa il relativo formato DOT, con il medesimo procedimento descritto in precedenza, da cui otterremo il file (*.gv*) contenente l'Unstructured Global Graph composto.

4.5 Fusione

Questa funzione è la più complessa e rappresenta il motivo principale per cui è stato appositamente sviluppato il tool. La fusione opera grazie a due tipi di input diversi, il primo è la struttura dati derivante da un grafo e il secondo è un insieme di etichette che definiscono i nodi che andranno fusi. La fusione è articolata in più fasi descritte di seguito.

Inizialmente vengono cercate tutte le combinazioni possibili di fusione tra i nodi e ogni coppia di nodi su cui è possibile eseguire l'operazione viene inserita in una lista contenente proprio le coppie degli *id* dei relativi nodi. Ottenendo di conseguenza un a lista di coppie di nodi destinati alla fusione.

Una volta ottenuta la lista di coppie da fondere, è possibile procedere con l'operazione vera e propria; viene generata una struttura temporanea che andrà a contenere gli spiders derivanti dalla fusione. Successivamente vengono cercati i nodi da fondere all'interno della struttura dati originale perché è necessario prelevare alcune informazioni fondamentali per la realizzazione degli spiders. È necessario infatti, per ogni nodo, prelevare dati riguardanti gli *id* dei nodi, gli archi entranti e gli archi uscenti. Contemporaneamente, nella struttura temporanea invece si crea, per ogni nodo da fondere, un "clone" avente lo stesso *id* ed archi entranti, cambia però, la sua etichetta che diventa una choice di apertura; viene creato anche un secondo nodo, sempre relativo al nodo da fondere, che rappresenta però, la chiusura della choice e ad esso vengono attribuiti gli archi in uscita del nodo originale. Questa fase rappresenta la creazione dei punti di connessione dello spider generato con il grafo originale. Per avere una idea generale si immagini che ad ogni nodo da fondere vengano sostituiti i due nodi di choice corrispondenti, dove però al loro interno possiedono una rete di archi e nodi più o meno articolata.

Viene successivamente creato il nodo di fusione vero e proprio, seguendo le regole concettuali introdotte all'interno del Background. A ognuno vengono collegati due nodi di parallel, rispettivamente uno in entrata ed uno in uscita. Facendo riferimento ai nodi di choice precedentemente creati; il parallel in entrata riceve gli archi in uscita dai choice in apertura, invece, il parallel in uscita manda un arco in ognuno dei choice di chiusura precedentemente creati. Si noti che questo non è possibile all'interno di uno Structured Global Graph. Nel caso in cui un nodo abbia più di una fusione possibile, dalla seconda volta, in cui viene richiamato per la fusione, in poi, non vengono più creati i due nodi di choice ma vengono semplicemente aggiunti degli archi uscenti dal primo e degli archi entranti nel secondo, rispettivamente in direzione e derivanti dal nodo di fusione. Questa fase conclude la creazione degli spiders.

Terminata la fase di fusione si cercano nuovamente i nodi modificati all'interno della struttura originale e si eliminano. La struttura temporanea viene completamente ricopiata all'interno della struttura originale, ottenendo così la fusione dei nodi all'interno del grafo. Infine, derivandone da essa il relativo formato DOT, otterremo il file (*.gv*) contenente l'Unstructured Global Graph con i nodi fusi tra loro.

4.6 Traduzione nelle reti di Petri

Per poter effettuare la traduzione, è stato necessario studiare e assegnare una corrispettiva rappresentazione attraverso le reti di Petri, a ogni tipo d'istruzione. Malgrado la presenza di queste regole, si è dovuto elaborare un nuovo metodo, condizionato da esse, per poter ottenere un algoritmo di traduzione. Innanzitutto, si è definita una nuova struttura dati (cod.6) che permettesse di riorganizzare il grafo prima della conversione. Anch'essa è data da un'array ed è la seguente.

```
class Petri:
    def __init__(self, id, im, trans, place, pin, *ist):
        self.id = id
        self.im = im
        self.tr = []
        self.pla = []
        self.p_in = []
        self.ist = []
        for h in range(trans.__len__()):
            self.tr.append(int(trans[h]))
        for i in ist:
            self.ist = i
        for k in range(place.__len__()):
            self.pla.append(int(place[k]))
        for j in range(pin.__len__()):
            self.p_in.append(int(pin[j]))
```

Codice 6: Struttura per traduzione in rete di Petri

Questa struttura dati ci permette di memorizzare, relativamente a ogni nodo, le seguenti informazioni: *id* univoco del nodo, *im* che esprime cosa rappresenta il nodo (ad esempio: istruzione, choice di apertura, etc...), *tr* array contenente gli *id* della transition, *pla* array contenente gli *id* delle place, *p_in* array contenente gli *id* delle place che hanno un arco entrante nella transition iniziale del nodo e infine *ist* contenente l'etichetta del nodo.

Per poter compilare la struttura dati delle reti di Petri nel modo corretto è necessario leggere due volte la struttura dati dei Global Graph; la prima volta per creare una corrispondenza di tutti i nodi e inserire le informazioni base, la seconda per completare i campi contenenti le informazioni relative agli archi. Vengono effettuate due scansioni perché per poter inserire i dati corretti relativi agli archi, è prima necessario avere tutti gli id dei nodi, delle place e delle transition. Non essendo possibile avere queste informazioni in anticipo è per l'appunto necessaria una doppia scansione. Grazie a quest'ultime informazioni sarà possibile infatti sfruttare le regole di trasformazione per la conversione.

Una volta completata la scrittura della struttura, per ottenere la traduzione, è sufficiente leggere il campo *im* di ogni nodo e poi in base alle informazioni contenute in *tr*, *pla* e in *p_in* applicare le regole studiate elencate di seguito.

Tuttavia, l'algoritmo che permette la scrittura del file non implementa le regole descritte all'interno del Background, ma ne utilizza una versione appositamente modificata per rendere più semplice la scrittura del codice. All'interno del campo *im* è possibile riconoscere otto tipi diversi d'istruzione. In base al contenuto di *im* si eseguiranno diverse operazioni; esso sono:

- "*start*" tradotto semplicemente con una place alla quale viene assegnato un token;
- "*istruction*" tradotta con una transition ed una place collegate da un arco;
- "*open choice*" tradotta con due transition differenti dove da ognuna esce un arco indirizzato ad una place, anch'esse differenti;
- "*transition choice*" tradotta con una transition ed una place collegate da un arco;
- "*close choice*" tradotta con due transition differenti aventi ciascuna un arco indirizzato in una place comune;
- "*open parallel*" tradotta con una transition da cui escono due archi indirizzati a due place differenti;
- "*close parallel*" tradotta con una transition ed una place collegate da un arco;
- "*end*" che viene ignorato e non viene rappresentato.

Gli *id* attribuiti alle place e alle transition sono ricavati dai campi *tr* e *pla*. Ogni qual volta che viene scritta la traduzione della rappresentazione di un nodo si analizza il campo *p_in* per poter creare e tracciare le giuste connessioni tra i nodi che precedono e il nodo appena rappresentato.

Una volta stabilite queste regole è stato sufficiente scriverle replicando il formato di scrittura dei file che utilizza il tool TINA (*.ndr*) per poterne ottenerne uno identico, corrispondente al Global Graph dato in input, e compatibile con il tool sopra citato.

Vediamo, nella pagina seguente un esempio concreto del lavoro di traduzione svolto da Domitilla. Dopo aver ottenuto il file *.gv* contenente la traduzione è sufficiente caricarlo all'interno del tool TINA per poter ottenerne la rappresentazione grafica (fig.12) ed eseguire tutte le modifiche che ci interessano. La traduzione è ottenuta avendo come input lo Structured Global Graph rappresentato in fig.7.

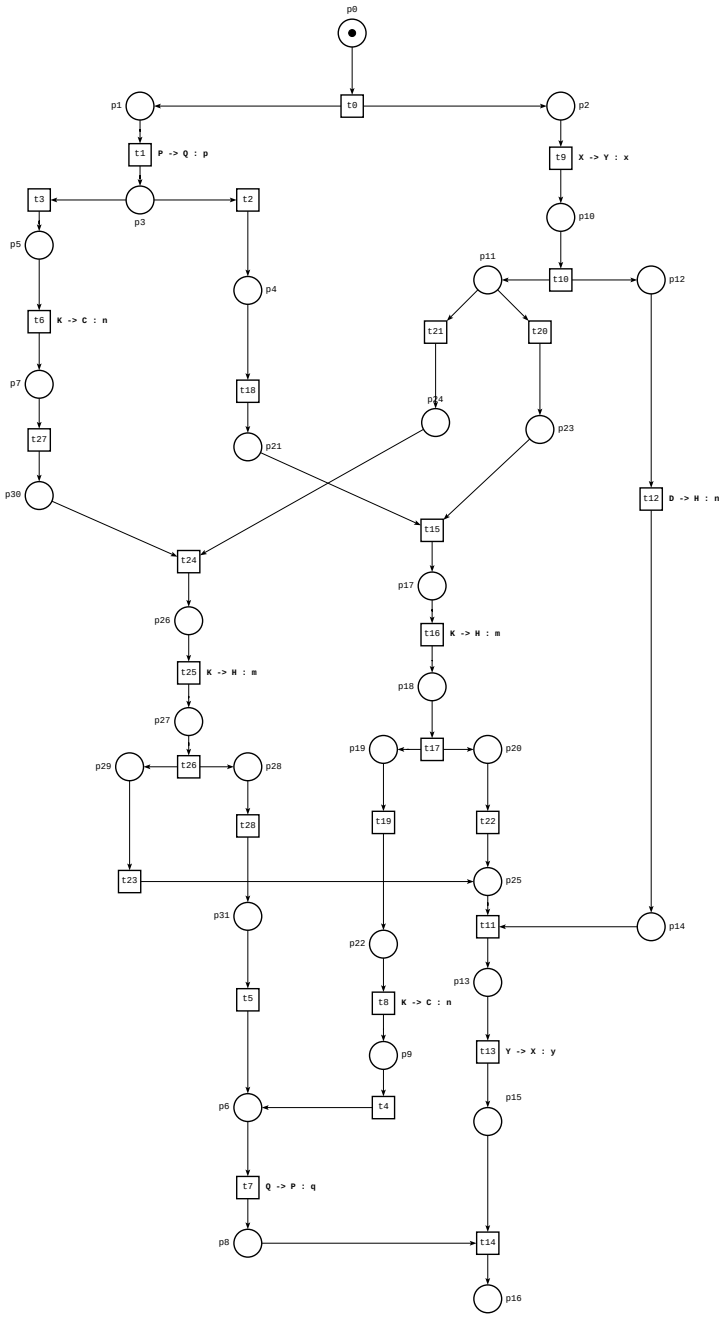

Figura 12: Prodotto della traduzione

5 Conclusioni

5.1 Sviluppi futuri

In futuro, Domitilla, potrà evolversi e continuare il suo percorso di completamento d'integrazione con TINA o con altri tool, per l'analisi dei grafi tradotti; sarebbe infatti molto utile, alla ricerca in corso e non solo, l'ottenimento dei grafi di raggiungibilità derivanti dai Global Graph. TINA già offre questa feature, quindi in futuro sarà possibile estendere Domitilla permettendole di leggere e di interpretare anche questo nuovo tipo di grafo.

Necessita, inoltre di una ottimizzazione del codice, per la realizzazione del tool non si è tenuto in considerazione la complessità o il costo degli algoritmi. Anche l'interfaccia grafica, al momento molto minimale ed essenziale, necessita di miglioramenti.

5.2 Conclusioni personali

Sono soddisfatto del risultato conseguito da questo progetto, l'obbiettivo è stato completamente raggiunto, nonostante ci siano ancora molte cose ancora da fare ed eventualmente aggiungere. Il problema legato alla non possibilità di rappresentare mediante Structured Global Graph la loro fusione è stato ampiamente risolto e superato; infatti non solo abbiamo ottenuto la conversione in modo automatico in Unstructured Global Graph grazie alla quale possiamo rappresentare la fusione dei nodi di un grafo ma siamo stati in grado di ottenere anche una traduzione nelle reti di Petri, che permette di eseguire ulteriori analisi e approfondimenti sui grafi.

6 Bibliografia

Riferimenti bibliografici

[1] Franco Barbanera, Ugo de'Liguoro, and Rolf Hennicker. Global types for open systems. In Bartoletti and Knight [2], pages 4–20.

[2] Massimo Bartoletti and Sophia Knight, editors. *Proceedings 11th Interaction and Concurrency Experience, ICE 2018, Madrid, Spain, June 20-21, 2018*, volume 279 of *EPTCS*, 2018.

[3] Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty session types meet communicating automata. In Seidl [13], pages 194–213.

[4] DOT the dot language. `https://graphviz.gitlab.io/_pages/doc/info/lang.html`. Accessed: 2019-02.

[5] Emden Gansner, Eleftherios Koutsofios, and Stephen North. Drawing graphs with dot. Technical report, 2006.

[6] Graphviz graph visualization software. `https://www.graphviz.org/`. Accessed: 2019-02.

[7] Hans Hüttel, Ivan Lanese, Vasco T. Vasconcelos, Luís Caires, Marco Carbone, Pierre-Malo Deniélou, Dimitris Mostrous, Luca Padovani, António Ravara, Emilio Tuosto, Hugo Torres Vieira, and Gianluigi Zavattaro. Foundations of session types and behavioural contracts. *ACM Comput. Surv.*, 49(1):3:1–3:36, 2016.

[8] JetBrains the drive to develop. `https://www.jetbrains.com/`. Accessed: 2019-02.

[9] Julien Lange, Emilio Tuosto, and Nobuko Yoshida. A tool for choreography-based analysis of message-passing software. In Simon Gay and António Ravara, editors, *Behavioural Types: from Theory to Tools*, chapter 6. River Publishers, 2017.

[10] Wuxu Peng. Deadlock detection in communicating finite state machines by even reachability analysis. In *Proceedings of Fourth International Conference on Computer Communications and Networks - IC3N'95*, pages 656–662, Sep. 1995.

[11] Carl Adam Petri. *Communication with automata*. PhD thesis, Universität Hamburg, 1966.

[12] Pycharm the python ide for professional developers. `https://www.jetbrains.com/pycharm/`. Accessed: 2019-02.

[13] Helmut Seidl, editor. *Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7211 of *Lecture Notes in Computer Science*. Springer, 2012.

[14] TINA time petri net analyzer. `http://projects.laas.fr/tina/papers.php`. Accessed: 2019-02.

Ringraziamenti

Vorrei ringraziare il Professor Lanese per avermi dato la possibilità di partecipare, seppur in minima parte, a questa ricerca ancora in corso, concedendomi di realizzare un tool che la supportasse.

Non è stato semplice, soprattutto all'inizio, perché, trattandosi per l'appunto di una ricerca in fase d'opera, la disponibilità di materiale su cui lavorare è esigua, ma il Professor Lanese è sempre stato disponibile e pronto per chiarimenti e delucidazioni; fornendomi talvolta porzioni di articoli non ancora pubblicati.

Vorrei ringraziare, inoltre, la mia famiglia che mi ha concesso la possibilità d'intraprendere questo percorso sostenendomi economicamente e moralmente, infine per ultimi, ma non meno importanti, vorrei ringraziare anche tutti i miei amici e compagni di università che mi hanno aiutato a mantenere la concentrazione necessari per superare le difficoltà presentate dal corso e allo stesso tempo concedendomi momenti di svago al di fuori delle aule di studio.