Università degli studi di Milano-Bicocca

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Informatica

Strumenti testuali e grafici per la specifica delle reti di Petri

Supervisori
Prof.ssa Lucia Pomello
Dott. Luca Bernardinello

Relazione della prova finale di
Marco Previtali
Matricola: 704496
Email: m.previtali3@campus.unimib.it
Telefono: 340 - 3959775

Seduta Finale: 25 Novembre 2010
Anno Accademico 2009/2010

L'obiettivo di questa tesi, svolta all'interno dello stage di Metodi Formali, è stato quello di continuare lo sviluppo del linguaggio di specifica per le reti di Petri chiamato PetriNet.
Le reti di Petri sono uno dei metodi di rappresentazione matematica di sistemi distribuiti che permettono la modellazione di sistemi ad eventi discreti e di effettuare considerazioni matematiche su di essi. Le reti di Petri prendono il loro nome dal loro creatore, Carl Adam Petri, che le propose come strumento per la modellazione di sistemi concorrenti durante la sua tesi di Dottorato nel 1962.
Una rete di Petri è un grafo bipartito formato da due elementi; i primi sono i Posti che rappresentano condizioni o stati locali del sistema e che vengono rappresentate tramite dei nodi circolari mentre i secondi elementi sono le Transizioni che rappresentano azioni effettuate dal sistema che possono venir eseguite se e solo se tutti gli stati collegati in ingresso sono attivi. Un Posto può contenere o meno una o più marche che vengono rappresentate con dei pallini neri all'interno del nodo e che rappresentano solitamente il fatto che un posto sia considerato attivo all'interno del sistema generale. Il numero di marche massimo che può essere contenuto da un posto prende il nome di capacità dell'elemento.
Posti e transizioni sono collegati tramite degli archi direzionati a cui può venir associata una funzione di peso che modifica il modo in cui ogni arco influisce sul numero di marche presente nel posto a cui è collegato. Nel caso in cui l'arco sia direzionato verso il posto il numero di marche, una volta scattata la transizione dalla quale l'arco parte, aumenterà in modo solidale con il peso. Nel caso in cui, invece, l'arco abbia verso uscente dal posto, una volta scattata la transizione nella quale l'arco entra, verranno tolte un numero di marche equivalente al peso.
Il lavoro svolto nel durante il mio stage ha apportato modifiche sostanziali al formalismo sviluppato e alle funzionalità fornite dal parser del linguaggio. Una delle prime modifiche apportate è stato il cambiamento delle reti ge-

stibili dal sistema. Precedentemente era possibile definire reti elementari, formate cioè da posti di capacità massima unitaria e peso degli archi limitato a uno. La nuova versione permette di gestire le più generiche reti P/T nelle quali le capacità dei posti sono illimitate e il peso degli archi deve essere un numero intero positivo qualsiasi. Come è facilmente intuibile le reti elementari sono un sottoinsieme proprio delle reti P/T. Seppur la generalizzazione di reti trattate possa sembrare una modifica relativamente semplice ciò ha implicato la necessità di riscrivere quasi tutto il modulo software che gestisce le strutture dati utili al funzionamento del sistema. Questo perché le logiche di funzionamento non sono completamente compatibili e perché gestendo reti elementari è possibile fare degli assunti non veri per le reti P/T.
Avendo modificato la classe di reti trattate ha portato anche a modificare la sintassi per la dichiarazione dei posti (in quanto deve essere possibile specificarne la capacità) e la sintassi della definizione dei flussi all'interno della rete (in quanto deve essere possibile specificare il peso degli archi).
Altra introduzione portata al linguaggio è la possibilità di gestire array monodimensionali di posti, transizioni e reti. Questa funzionalità è molto utile perché permette di indicizzare degli elementi e, utilizzando il costrutto `for` del quale si discuterà in seguito, permette di diminuire la quantità di codice necessaria alla definizione delle reti.
Come appena accennato un'altra aggiunta apportata al linguaggio è stata l'introduzione di un costrutto atto a effettuare ciclicamente una serie di comandi. Questo costrutto è stato progettato seguendo il formato del linguaggio C in quanto prevede una sintassi molto conosciuta in ambito informatico e permette di far assimilare il concetto facilmente.
Introduzione di minore impatto sulla funzionalità del linguaggio ma comunque molto utile a livello espressivo è stata l'aggiunta della possibilità di inserire commenti all'interno del codice da parte dell'utente per poter spiegare meglio ciò che il codice si prefigge di fare.
É stato scelto di definire una sintassi dei commenti simile, anche in questo

caso, al linguaggio C per gli stessi motivi per cui il ciclo for è stato implementato in modo analogo al linguaggio in questione.
Funzionalità molto importante inserita nel linguaggio PetriNet durante il mio stage è stata quella di permettere l'unione di reti precedentemente dichiarate. La possibilità di unire due reti era già presente nella versione precedente del sistema ma era limitata all'unione di due sole reti e venivano commessi degli errori durante questa procedura. Avendo portato questa funzione alla capacità di unire un numero arbitrario di reti ne ha aumentato la complessità da una parte ma ne ha migliorato la potenza e l'espressività dall'altra.
Durante lo sviluppo di PetriNet ci si è accorti che in certe situazioni è necessario permettere all'utente di specificare il comportamento del sistema durante la procedura di unione. In particolare il problema è sorto pensando al fatto di avere o meno nella nuova rete creata un prefisso per ogni elemento in modo da capire da quale rete l'elemento stesso deriva. Per poter venire incontro a queste necessità è stato deciso di permettere all'utente di influire sul comportamento del sistema inserendo delle *direttive al parser* simili alle direttive al preprocessore del linguaggio C.
Questo concetto di direttive al parser è stato riutilizzato inoltre per poter gestire il comportamento di default del sistema in alcuni casi quali la capacità assegnata ad un posto quando non definita.
É stata inoltre inserita la possibilità di esportare una rete in formato PNML (Petri Net Markup Language). PNML è un formato standardizzato di definizione delle reti di Petri basato su XML che viene utilizzato da diversi software di analisi e gestione. In questo modo è stato possibile svincolare il linguaggio da compiti di analisi che precedentemente erano stati inseriti che complicavano inutilmente il sistema e allontanavano PetriNet dalla sua natura di linguaggio di specifica di reti.
Quasi tutto il software sviluppato durante lo stage è stato scritto in Python poiché è un linguaggio molto potente che fornisce, tramite la sua completa libreria standard, diverse facilitazioni nella scrittura del codice. É inoltre

pensiero comune che Python abbia una curva di apprendimento molto bassa e che quindi si possa essere produttivi in poco tempo dall'inizio dello studio del linguaggio.
L'architettura del software si compone di 4 moduli Python e di uno script Bash.
PetriNet.py è il modulo che si occupa di definire le strutture dati utili a descrivere una rete di Petri e a gestirne le relazioni. Come scritto in precedenza questo modulo è stato sostanzialmente riscritto durante il mio lavoro per poter supportare la definizione di reti P/T.
PetriNet_lex.py è il modulo che si occupa di suddividere il codice passato al sistema in token e di effettuare l'analisi grammaticale degli elementi passati. Il suo funzionamento è abbastanza semplice, al suo interno vengono definite le regole di identificazione di token utilizzando delle espressioni regolari verificando che siano parole accettate dal linguaggio e fornisce il risultato all'esterno (nel nostro caso al parser). Questo modulo basa completamente il so funzionamento sul modulo `lex` della libreria `ply`.
PetriNet_parser.py è il modulo che si occupa di fare l'analisi sintattica del codice fornito. Al suo interno è possibile trovare le regole che definiscono come una descrizione di una rete di Petri può essere effettuata e basa completamente il suo funzionamento sul modulo `yacc` della libreria `ply`.
Per poter dotare il linguaggio del costrutto `for` è stato necessario sviluppare un quarto modulo chiamato pnpre.py che si occupa di preprocessare il file espandendo in modo corretto i cicli prima che parser e lexer svolgano i loro compiti.
Ultimo modulo sviluppato è stato uno script in linguaggio Bash chiamato `pnc` che permette di richiamare in modo corretto i comandi sul codice che si vuole processare.