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

The objective of this thesis, performed in the stage of Metodi Formali, was to continue the development of PetriNet a specification language for Petri nets.

Petri nets are one of the mathematical methods used to represent distributed-systems, to model discrete event systems and carry out mathematical considerations on them. Petri nets owe their name to their creator, Carl Adam Petri, who proposed this formalism as a tool for modeling concurrent systems during his PhD in 1962.

A Petri net is substantially a bipartite graph formed by two elements, the first are places used to represent local conditions or states of the system and are represented by circular nodes and the second element are transitions that represent actions performed by the system that can be executed if and only if all the places connected in the input are active. A place may contain zero or more marks. Marks are represented by blacks dots within the node, and typically represent the fact that a place is considered active in the system. The maximum number of marks that can be contained in a place is called capability.

Places and transitions are connected via directed arcs that can be associated to a weight function that changes the way in which each arch affects the number of marks present in the place connected. If the arc is pointing towards the place the number of marks will increase in the same amount of the weight of the arc when the transition connected fires. If the direction of the arch is leaving the place when the transition fires will be removed a number of marks equivalent to the weight.

The work done during my stage has made substantial changes to the formalism developed and the functionality provided by the language parser. One of the first changes is the change of networks that the system can handle. Previously it was possible to define only elementary networks, that are formed by places that have capacity limited to one and the weight of the arches is limited to one. The new version allows you to manage more generals P/T net-

works in which the capacity of places can be unlimited and the weight of the arches can be any positive integer. As is easy to understand the elementary networks are a proper subset of P/T networks. Although the generalization of treated nets may seem a relatively simple change that implied to rewrite most of the software module that manages data structures because the logic and operation are not fully compatible.

By altering the class of treated nets has also led to change the syntax for the declaration of places (because you have to specify their capability) and the syntax used to define flows within the network (because you have to specify their weight).

Another new feature to the language is that now is possible to manage monodimensional arrays of places, transitions and networks. This functionality is very useful because it allows indexing of the elements and, using the construct *for* which we will discuss later, can decrease the amount of code required to define a network.

As I just mentioned another addition was made to the language is the introduction of a construct able to make a cycle. This construct is inspired by the format of the C language because it provides a syntax known in computer science and allows you to assimilate the concept quickly.

Introduction of lesser impact on the functionality of the language is the ability to insert comments within the code to explain to what the code aims to do. The syntax of comments' style is the same as in C.

One of the most important feature that has been added to the language during my stage is the capability of the language to allow users to merge an arbitrary number of nets.

Merge two networks was already possible in the previous version of the system but wasn't correct. During the development of PetriNet we noticed that in certain situations, it is necessary to allow the user to specify the behavior of the system during the union process. In particular, the problem arose when thinking to allow or not the new network to created a prefix for each

element to figure out from which network elements derive. To meet these requirements I decided to allow the user to influence the behavior of the system by placing directives to the parser similar to the C language preprocessor directives.
This concept of directives to the parser is been reused to modify the default behavior of the system in some other cases for example to set the default capacity assigned to a place if not defined.
It has also been introduced a feature that allow users to export a network in PNML (Petri Net Markup Language). PNML is a standardized format used to define Petri nets based on XML that is used by different analysis and management software. In this way it was possible to release the language analysis tasks that previously had been included in the language.
Almost all software developed during the internship is written in Python because is a very powerful programming language that provides, through its powerful standard library, facilities to write the code.
Is furthermore common thought that Python has a very quick learning curve, and then you can be productive in a short time from when you begin to study of language.
The software architecture consists of four modules and a Python script Bash.
PetriNet.py is the module that is responsible to define data structures that handle the concept of a Petri net and manage relationships between the elements. As previously written in this form it was fundamentally rewritten during my work to support P/T networks.
PetriNet_lex.py is the module that takes care of dividing the code into tokens and analyze the grammatical correctness. Its operations are simple, inside this module were defined rules used to identify tokens using regular expressions and this tokens will be passed to the outside (in our case the parser). This module based on module operation completely know the lex library ply.
PetriNet_parser.py is the module that takes care to parse the cod. Inside it is possible to find the rules that define how a description of a Petri net can

be done and bases its operation on the module yacc of ply.
In order to provide the language construct for I developed a fourth module called pnpre.py that deals with expanding the file correctly before the parser and lexer perform their duties.
Last module developed is a Bash script language called pnc that allows you to in the correct way the controls on the code to be processed.