Simulador de redes de Petri implementado com *Python 3*

Renan S. Silva

[1]Departamento de Ciencia da Computação –
Universidade do Estado de Santa Catarina (UDESC)
Santa Cataria – SC – Brazil

uber.renan@gmail.com

Abstract. *This paper describes an implementation of a petri net simulator wrote with Python 3, made for a Formal Methods class. The simulator has some basic functions such as insertion and remotion of places, transitions and tokens both interactively and from a file.*

Resumo. *Este documento descreve a implementação de um simulador de Redes de Petri escrito utilizando Python 3, feito para a disciplina de Métodos Formais. O simulador deverá conter funções básicas como adição e remoção de lugares, transições e tokens de forma iterativa e a partir de um arquivo.*

1. Conectividade e pureza

O primeiro problema na qual o simulador deve ser capaz de resolver é detectar se uma rede de Petri G é conexa. Dadas as características de um uma rede de Petri qualquer, existe apenas a necessidade de que a rede seja fracamente conexa. A rede ser fortemente conexa não é um pré requisito, portanto a execução de um *Depth First Search* (*DFS*) tratando a rede de Petri como sendo um grafo não direcionado seguido da contagem dos vértices alcançados é suficiente para determinar a conectividade da rede. No entanto foi utilizado um algoritmo mais simples, um algoritmo de *Roy* modificado para testar se um grafo é conexo, forte ou fracamente.
O segundo problema se trata de identificar se uma rede é pura ou impura. Isto é realizado de maneira muito simples. Dado um arco no formato $t_1 \rightarrow p_1$, é verificado se existe um arco $p_1 \rightarrow t_1$, caso exista, a rede é impura.
No console do simulador os comandos *test connectivity* e *test purity* são capazes de realizar as operações acima.

2. Matrizes de incidência, pré e pós condição

Dado uma rede de Petri qualquer é possível fazer sua representação matricial. Foram implementadas funções que retornam as matrizes de pré (I) e pós condição (O) e uma terceira função que retorna a matriz de incidência $C = O - I$. Dois dicionários mapeiam os lugares para linhas e as transições para colunas, de tal forma que dado uma transição $p_1 \rightarrow t_1\omega = x$, onde ω é o peso, é possível mapear um arco para a matriz. Iterando por todos os arcos criamos as matrizes I e O, dependendo se a transição é da forma $p_1 \rightarrow t_1$ ou $t_1 \rightarrow p_1$, respectivamente.
Para obter o vetor m_n com os *tokens* é utilizado também um dicionario para mapear as transições para as posições do vetor.
As funções *print precondition*, *print poscondition* e *print incidence* imprimem as matrizes

I, O e C respectivamente.
É possível imprimir o vetor de *tokens* usando o comando *print token vector*. Ao usar o comando *print tokens* sera exibido uma lista de todos os lugares que possuem 1 ou mais *tokens* e o seu respectivo numero de *tokens*.

3. Disparo de transições

Uma a função *trigger* t_n dispara a transição t_n se a mesma estiver ativa. A verificação da possibilidade de disparos é feita utilizando a matrizes de pré-condição junto com o vetor de *tokens*. Dado uma coluna referente a uma transição T_n, a transição esta habilitada se $\forall\, p_i \in P, M(p_i) - I(p_i, T_n) \geq 0$.
O comando *trigger* $t_1 t_2 \ldots t_n$ executará as transições passadas como argumento até que alguma delas não esteja ativa ou todas tenham sido disparadas. Para ver o resultado após o disparo basta usar o comando *print token matrix*.

4. Remoção e inserção dinâmica

É possível inserir lugares $p_1 p_2 \ldots p_n$ utilizando o comando *insert places* $p_1 p_2 \ldots p_n$. Para a inserção de transições $t_1 t_2 \ldots t_n$ é utilizado o comando *insert trans* $t_1 t_2 \ldots t_n$. A inserção de um arco $t_a \rightarrow p_a$ ocorre somente se $t_a \in T$ e $p_a \in P$.

5. Comandos

insert places p1 p2 . . . ***pn*** Insere os lugares p1 p2 . . . pn. Duplicatas são ignoradas
insert trans t1 t2 . . . ***tn*** Insere as transições t1 t2 . . . tn. Duplicatas são ignoradas
insert arc p1 -> t1 Insere um arco de p1 para t1, se e somente se t1 inT e p1 inP ou vice versa.
insert arc p1 -> t1 n Insere um arco de p1 para t1 com peso n. Sujeito as restrições do item acima.
set token pn w Muda o *token* do lugar pn para o peso w.
print places Imprime os lugares.
print trans Imprime as transições.
print edges Imprime os arcos.
print tokens Imprime os lugares que possuem um *token* ou mais.
print token vector Imprime o vetor de *tokens*.
print Imprime todas as informações relativas a rede.
test tn Retorna *True* ou *False* caso a tn esteja habilitado ou não.
trigger t1 t2 . . . ***t2*** Dispara as transições t1 t2 . . . t2.
test connectivity Retorna *True* ou *False* caso a rede seja conexa ou não.
remove p1 Remove p1 da rede. Funciona tanto para lugares quanto para transições e arcos.
test purity Retorna *True* ou *False* caso a rede seja pura ou impura.
print prec Imprime a matriz de pre condição.
print proc Imprime a matriz de pós condição.
print incidence Imprime a matriz de pre incidência.
close Fecha o programa.