Simulador (motor) de Redes de Petri

I. Introdução

O programa deverá permitir a montagem e execução de uma Rede de Petri do tipo *Lugar-Transição ~~Temporizadas~~* (*~~Timed~~ Place-Transitions* Nets).

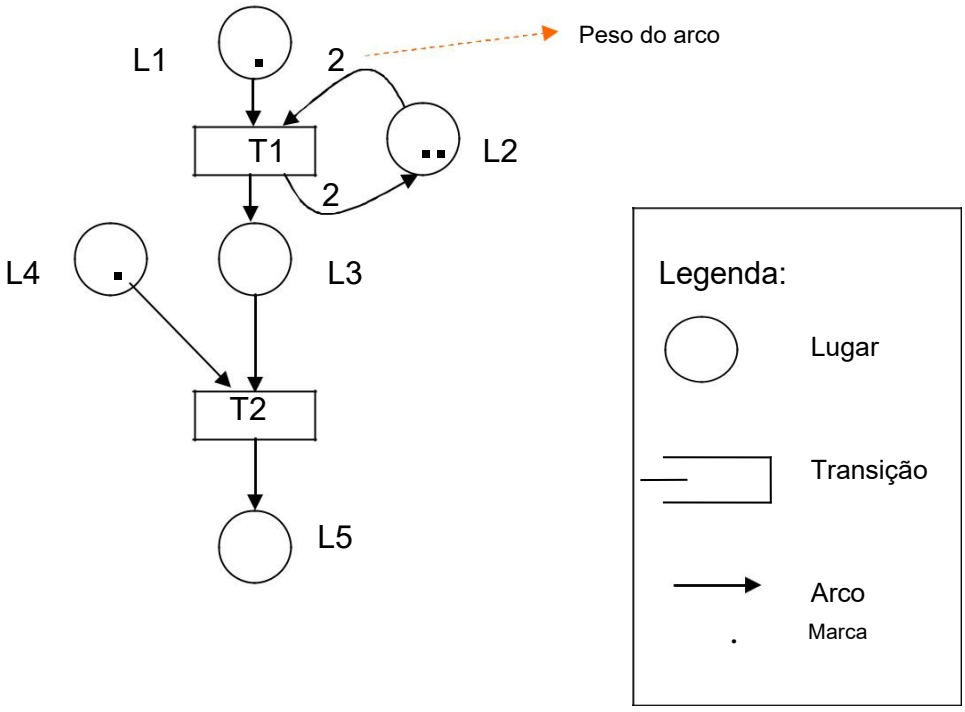
A forma como uma rede pode ser montada, manipulada e executada pode se dar de diferentes formas:

- de maneira interativa: com o usuário, permitindo que o usuário defina a quantidade de lugares e o número de transições que a rede conterá, bem como a marcação inicial (quantidade de marcas em cada lugar da rede) e o peso dos arcos que unem lugares e transições;
- a partir da leitura/carga da descrição de uma rede: por exemplo, a partir da leitura de um arquivo que contém a descrição da rede (o formato PNML é um exemplo de formato de representação voltado para Redes de Petri);
- através da criação e uso de uma API que exponha os métodos para criação, manipulação e execução da rede.

Quando da execução (simulação) da rede, esta execução deverá poder ser passo-a-passo, e a cada ciclo de execução deverá ser mostrado quais as transições habilitadas para o próximo ciclo e o número de marcas em cada um dos lugares da rede.

Não é necessário o uso de interface gráfica; a apresentação dos resultados pode ser feita toda via console, na forma de tabelas.

Exemplo de Rede de Petri:



Obs.: na rede da página anterior, no arco que vai de L2 para T1, o valor (peso) 2 indica que é necessário consumir 2 marcas de L2 para disparar T1 (desde que seja consumida também uma marca de L1); este valor é denominado de **Peso do Arco**.

Comentários:

1. a saída da execução do programa é textual; pode-se apresentar a informação através de tabelas. Exemplo:

Lugar	1	2	3	4	5
Marcação	1	2	0	1	0

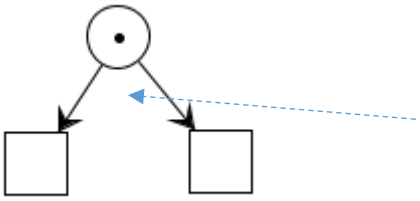
Transição	1	2	3
Habilitada ?	S	N	N

2. Critério para definir o disparo de transições: deve haver marcas **suficientes** em TODOS lugares de entrada de uma transição para esta estar habilitada. Quando uma transição for disparada, as marcas são consumidas dos seus lugares de entrada e são enviadas marcas para seus lugares de saída. A quantidade de marcas consumidas e enviadas depende do peso dos arcos que conectam lugares e transições. Um arco sem valor associado tem peso=1.
3. Cada ciclo de execução corresponderá a execução do disparo de todas transições habilitadas; o número do ciclo deverá ser mostrado na tela; para passar de um ciclo a outro (isto é, avançar a simulação) o usuário deverá apertar a tecla ENTER.
4. Caso a abordagem da aplicação seja interativa, no início da execução do programa o usuário poderá fornecer os dados necessários para a montagem da rede de forma interativa (ou através da leitura de um arquivo que contenha esta descrição). Exemplo:

   Quantos lugares: **3** Quantas
   transições: **2**
   Quais são os lugares de entrada de T1? **1, 3** Quais são os
   lugares de entrada de T2? **2, 3** Quantas marcas em L1 ?
   10
   Quantas marcas em L2 ? **4** Quantas
   marcas em L3 ? **0**
   Qual o peso do arco de L1 para T1 ? **1** Qual o peso
   do arco de L3 para T1 ? **2**
   :

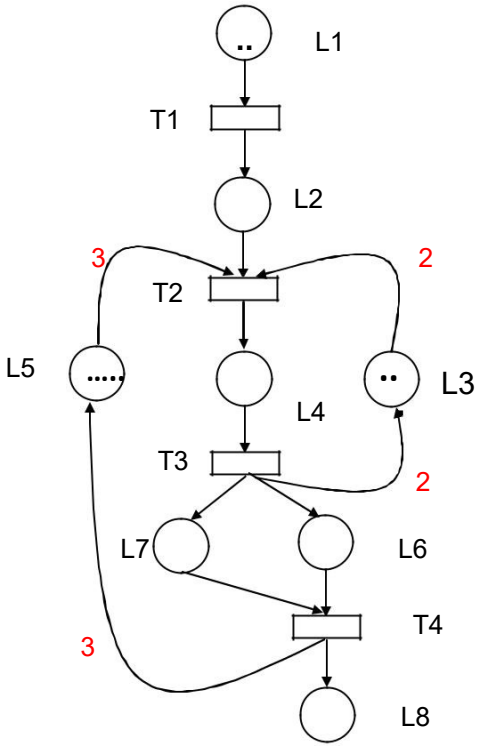
   Os dados necessários para a **montagem da rede** podem também ser fornecidos através da leitura de um arquivo que contenha esta descrição ou então de maneira procedural através de chamadas de métodos oferecidos pelo motor. Obs.: Caso se opte pela leitura de um arquivo contendo a definição da rede, pode-se empregar o formato PNML, que é um formato padrão para armazenamento de redes de Petri. Referências:
   a. http://www.pnml.org/
   b. https://www.wikiwand.com/en/Petri_Net_Markup_Language
   c. http://pnml.lip6.fr/
   d. http://xml.coverpages.org/pnml.html

5. **Resolução de concorrência**: quando a partir de um determinado lugar parte (sai) mais de um arco, o destino da marca (token) e' indeterminado. Cabe a *engine* escolher qual transição de saída será alimentada. Esta escolha será feita em função do lugar ou do token.



A *engine* pode requisitar a decisão para o lugar ou para o token que esta presente neste lugar. Isto pode ser feito atraves de um metodo disponibilizado pelo Lugar ou pelo token e que pode ser invocado pela *engine*; o valor de retorno deste metodo indicara a transição que será efetivamente habilitada, resolvendo deste modo o conflito.

II. Exemplo de rede e execução:

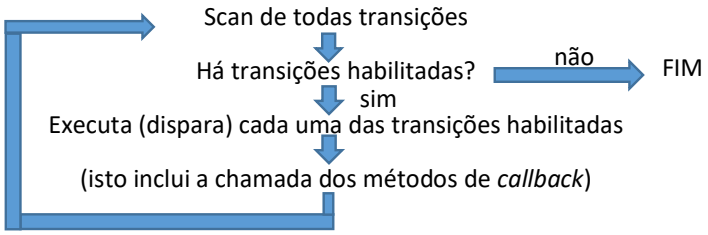


Quantidade de marcas em cada lugar	Transição habilitada ?

Núm. do ciclo	L1	L2	L3	L4	L5	L6	L7	L8	T1	T2	T3	T4
0 (inicial)	2	-	2	-	5	-	-	-	S	N	N	N
1	0	2	2	-	5	-	-	-	N	S	N	N
2	-	1	-	1	2	-	-	-	N	N	S	N
3	-	1	2	-	2	1	1	-	N	N	N	S
4	-	1	2	-	5	-	-	1	N	S	N	N
5	-	-	-	1	2	-	-	1	N	N	S	N
6	-	-	2	-	2	1	1	1	N	N	N	S
7	-	-	2	-	5	-	-	2	N	N	N	N

III. Arquitetura do Simulador/motor

Ciclo de execução da engine RdP:



Obs.: a regra para determinar se uma transição está habilitada é se há marcas (isto é, tokens) suficientes (de acordo com o peso de cada arco de entrada) em todos lugares de entrada da transição. Quando a transição e' executada (disparada) a transição consome as marcas dos lugares de entrada (de acordo com o peso de cada arco)e gera marcas (tokens) em todos lugares de saida (de acordo com o peso de cada arco saida) .

Obs.2: pode ser interessante associar alguns eventos com métodos de ***callback***; por exemplo, a ocorrência de um evento como o disparo de uma determinada transição pode ocasionar a chamada de um método de callback que irá executar um método externo, no escopo da aplicação.

Sugestão de classes

- ~~***Token*** (representa uma marca; opcionalmente, pode conter atributos)~~
- ***Lugar*** (armazena marcas)
- ***Transição***
- ***Conexão (ou Arco de conexão)*** de lugar para transição ou de transição para lugar; tem atributo peso; pode ser do tipo *normal*, *inibidor* ou *reset*
- ***RedeDePetri*** contendo uma rede (grafo) de lugares e transições

Sugestão de métodos para uma API

Criando/editando a geometria da rede

- boolean ***criaLugar***(int *id*) *// id e' a identificação do lugar ou transição*
- Lugar ***getLugar***(int *id)*
- boolean ***removeLugar***(int *id*)
- boolean ***criaTransicao***(int *id)*
- Transicao ***getTransicao***(int *id)*
- boolean ***removeTransicao***(int *id*)
- boolean ***criaConexao***(Lugar *lugar*, Transicao *transicao*, int *peso*, boolean *ehEntrada*, boolean *ehArcoInibidor*, *boolean ehArcoReset*)
- boolean ***removeConexao***(Lugar *lugar*, Transicao *transicao*)
- Lugar ***getLugarDeConexao***(Conexao conexao)
- Transicao ***getTransicaoDeConexao***(Conexao conexao)
- Conexao[] ***getConexoesEntrada***(int *id)* // retorna array de conexões de entrada de uma transição
- Conexao[] ***getConexoesSaida***(int *id)* // retorna array de conexões de saida de uma transição

Alterando/inspecionando a rede

- ~~void ***insereTokenEmLugar***(Token *token*, Lugar *lugar*)~~ → void ***addTokens***(int *quantity*, Lugar *lugar*)
- boolean ***removeTokenDeLugar***(Token *token*, Lugar *lugar*)
- void ***clearLugar***(Lugar *lugar*) // remove todos tokens do lugar

- Token ***getToken***(Lugar *lugar*) //retorna token
- Token[] ***getToken***(Lugar *lugar*) //retorna um array de tokens
- int ***quantosTokens***(int *id*) //retorna a quantidade de tokens de um lugar com este id

- boolean ***getStatusTransicao*** (int *id*) // retona True se Transição habilitada e False caso contrário;
- void ***setTransicaoInativa***(int *id*) // seta transicao como inativa
- void ***setTransicaoAtiva***(int *id*) // seta transicao como ativa (default)
- boolean ***isTransicaoAtiva***(int *id*)

- boolean ***salvaRede*** (String *nomeArquivo*)
- Rede ***carregaRede*** (String n*omeArquivo*)

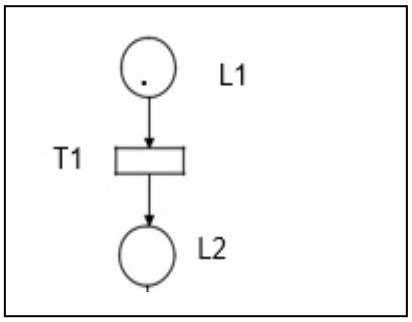
métodos opcionais

Simulando/executando/conversando com o ambiente externo

- void ***executaCiclo***(), varre toda a rede, identificando todas transições habilitadas e executando cada uma destas transições habilitadas; a movimentação de tokens e o disparo de transições podem acarretar a chamada de métodos de *callback* para a camada de visualização/interação.
- boolean ***insereCallbackTokenEntrandoLugar***(Lugar *lugar*, *ponteiroPara Método/Função*, Token *token):* método da aplicação é invocado (enviando token como argumento) se um token é inserido em um lugar;
- boolean ***insereCallbackTokenSaindoLugar***(Lugar *lugar*, *ponteiroPara Método/Função*, Token *token):* método da aplicação é invocado (enviando token como argumento) se token é removido de um lugar;
- boolean ***insereCallbackTransicao***(Transicao transicao, *ponteiroPara Método/Função) método da aplicação é invocado se transição é disparada.*

Obs.: podem ser criados outros métodos adicionais, conforme necessidade.

Exemplo de uso da API:




```
RedeDePetri  rede = new RedeDePetri ();

Token t1 = new Token();
```

```
rede.criaLugar(0);   // L1
rede.criaLugar(1);   // L2
rede.criaTransicao(0);    // T1
rede.criaConexao(rede.getLugar(0), rede.getTransicao(0), 1, true, false);  //conecta lugar de entrada
rede.criaConexao(rede.getLugar(0), rede.getTransicao(0), 1, false, false); //conecta lugar de saida

rede.insereTokenEmLugar(t1, rede.getLugar(0));
rede.executaCiclo();
```