Escola de Engenharia
Universidade do Minho

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
Mestrado Integrado em Engenharia Informática
Arquitetura e Cálculo

Trabalho Prático

TP1

Manuel Monteiro - A74036

Tiago Baptista - A75328

Braga, 18 de Maio de 2020

Conteúdo

1. Introdução

Este trabalho surge no âmbito da unidade curricular de Arquitetura e Cálculo pertencente ao perfil de Métodos Formais em Engenharia de Software. Este relatório visa expor o problema e a abordagem da resolução do mesmo.

Este problema consiste em quatro aventureiros que querem passar uma ponte, mas esta só aguenta com o peso de dois de cada vez e além disso é necessário levarem uma lanterna para fazer a passagem. Além disso, os aventureiro não têm a mesma capacidade para atravessarem a ponte sendo que o mais rápido demora 1 minuto e o mais lento 10. Os restantes demoram 2 e 5 minutos. Para resolver, foi feito um sistema em tempo-real com base na ferramenta UPPAAL.

2. Estrutura

Foram definidos os dois sistemas: dos aventureiros e o da lanterna. Além disso, foram definidas 4 ações sincronizantes a serem utilizadas em ambos os sistemas: **Pegar/Largar** e **Acompanhar/Desacompanhar**. Uma variável guarda o lado em que a lanterna se situa (*L*), outra armazena o portador da lanterna (*portador*) e por fim um relógio que irá contar o tempo total do sistema. Os sistemas são depois corridos em paralelo.

De seguida iremos explicar cada um dos sistemas e como foram tratados.

2.1 Aventureiro

Cada aventureiro começa no lado inicial e recebe o seu tempo de travessia e o seu identificador como argumentos. Depois, para a primeira travessia de cada um, testamos se a lanterna está no lado inicial ($L == 0$) e se ninguém se encontra a segurar a lanterna ($portador == 0$) para efetuar a ação **pegar!** e damos *reset* ao relógio que conta o tempo individual do aventureiro e alteramos o *portador* para o identificador do aventureiro que tomou essa ação. Se a lanterna já estiver a ser segurada, deixamos ainda que ocorra outra ação, **acompanhar**, pois podem 2 aventureiros atravessar a ponte de cada vez. De seguida fazemos a ação **desacompanhar!**, caso o aventureiro não seja o portador da lanterna e contamos o tempo de travessia do mesmo. Caso seja o portador da lanterna, a ação efetuada é a de **largar!**, contamos o tempo com a guarda, tendo assim o tempo de travessia do aventureiro, como também atribuímos o valor 0 à variável *portador* indicando que já ninguém a está a segurar.

Para a solução do problema é necessário que alguns aventureiros voltem para o lado inicial, portanto os mesmos podem **pegar!** na lanterna outra vez (caso esta esteja no lado certo e não esteja a ser segurada), dando **reset** ao relógio individual para contar o tempo de travessia outra vez, levando a mesma para o lado inicial, para repetir o processo.

Por fim fazemos o reparo que para a solução do problema, não nos fez sentido haver um acompanhante para voltar ao lado inicial, daí essa ação só se verifique do lado inicial para o lado final da travessia.

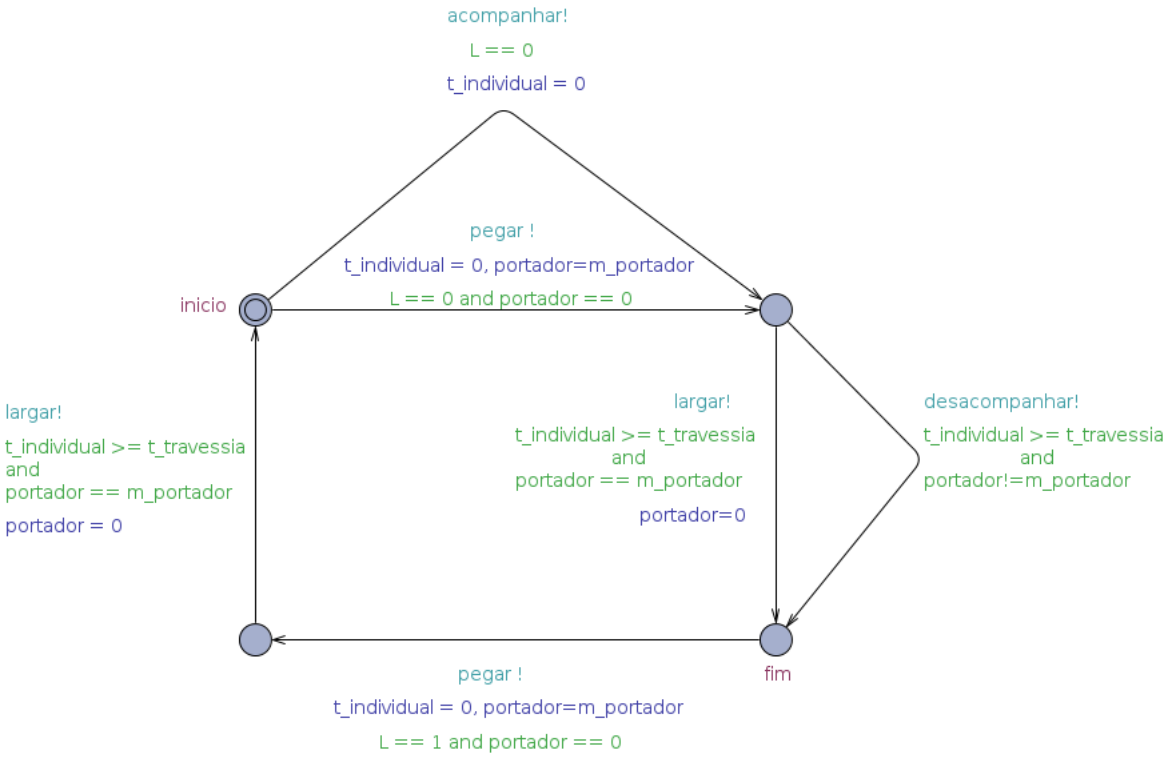

Figura 2.1: *Template* do aventureiro

2.2 Lanterna

A lanterna começa no estado livre e de seguida pode evoluir para o estado um, que significa que um aventureiro efectuou a acção de **pegar?** na lanterna e iniciou a travessia da ponte sozinho ou então para o estado dois, que significa que após um aventureiro estar a segurar a lanterna o outro efectua a acção de **acompanhar?** para os dois atravessarem a ponte.

De maneira a ser possível modelar este comportamento optamos pelo uso de uma localização intermédia *urgent* de maneira a que não ocorra passagem de tempo entre o portador da lanterna começar a atravessar sozinho (evoluir para o estado um), ou outro aventureiro juntar-se à travessia, efectuando a acção *acompanhar* (evoluir para o estado dois).

Para controlo do lado em que o aventureiro se encontra é efectuada um update na variável de controlo do lado (o *L*).

Tomando um exemplo do comportamento, um aventureiro começa por efetuar a ação **pegar!**, ação esta que é efetuada de seguida por um outro aventureiro inciando-se assim a travessia. De seguida ambos os aventureiros ficam bloqueados no estado e apenas podem passar para o estado fim quando o seu tempo individual de travessia (que começa a ser contado quando a travessia for iniciado) for superior ao seu tempo de travessia fixo e definido no enunciado.Quando o tempo de travessia do portador termina

é efetuada a acção de *!largar* e quando o tempo do acompanhante termina é efetuada a ação de *!desacompanhar*. De seguida um dos viajantes que chegou ao lado de fim pode iniciar a travessia no sentido contrário de maneira a levar a lanterna ao outro lado para que os próximos viajantes possam começar a sua travessia.

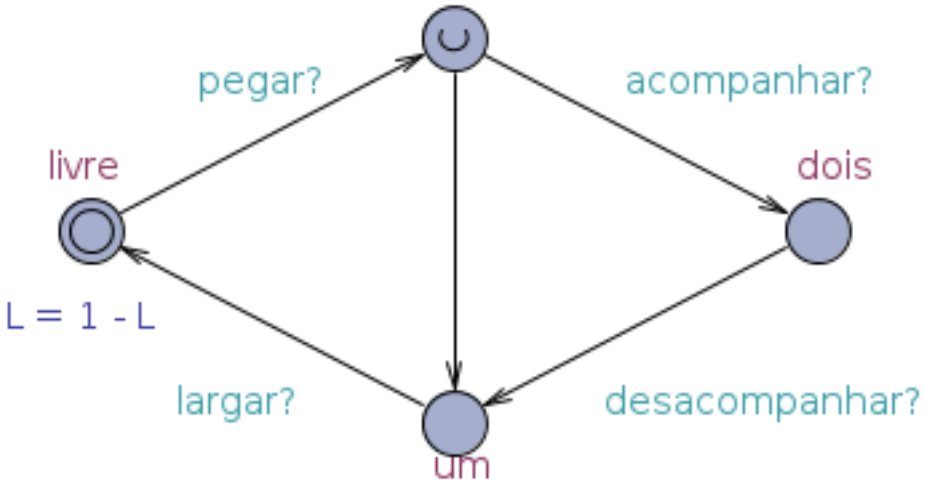

Figura 2.2: *Template* da Lanterna

3. Propriedades expressivas

Para o sistema ficar completo, tentou-se verificar o uso de certas propriedades para garantir que era possível que o sistema evoluísse da forma pretendida. Esta secção visa demonstrar e explicar as propriedades codificadas em CTL.

3.1 *Safety*

- **A [] not deadlock**

  Esta propriedade serve para garantir que o sistema não entra num estado em que não pode sair ficando assim bloqueado.

- **A[] not (Aventureiro4.fim and relogio<10)**

  Esta propriedade garante que o aventureiro mais lento ao atravessar, o tempo total não será menor do que o tempo de travessia do mesmo.

3.2 *Liveness*

- **A<> not (Aventureiro1.inicio or Aventureiro2.inicio or Aventureiro3.inicio or Aventureiro4.inicio) imply portador>0**

  Esta propriedade garante que se um dos aventureiros não estiver no estado inicial então o algum aventureiro estará a segurar na lanterna

3.3 *Reachability*

- **E <> Aventureiro1.fim**

  Esta propriedade serve para garantir que o sistema tem um caminho em que eventualmente o Aventureiro1 está no fim.

- **E <> Aventureiro2.fim**

  Esta propriedade serve para garantir que o sistema tem um caminho em que eventualmente o Aventureiro2 está no fim.

- **E <> Aventureiro3.fim**

  Esta propriedade serve para garantir que o sistema tem um caminho em que eventualmente o Aventureiro3 está no fim.

- **E <> Aventureiro4.fim**

  Esta propriedade serve para garantir que o sistema tem um caminho em que eventualmente o Aventureiro4 está no fim.

- **A[] not (Aventureiro4.fim and relogio<10)**

  Esta propriedade serve para verificar que em todos os *traces* o tempo de travessia tem de ser maior que o tempo de travessia do aventureiro mais lento.

- **E<> Aventureiro4.fim imply relogio>=10**

  Esta propriedade serve para verificar que se o aventureiro 4 se encontrar no fim, vai ter passado pelo menos 10 minutos no relógio global do sistema.

4. Resultados

Como resultado apresentam-se as fórmulas pedidas para serem codificadas em CTL e a sua satisfazibilidade.

- **E<> Aventureiro1.fim and Aventureiro2.fim and Aventureiro3.fim and Aventureiro4.fim and relogio==17**

  Esta propriedade codifica em *CTL* a possibilidade de todos os aventureiros passarem de um lado para o outro em 17 minutos.Obteve-se a satisfazibilidade da fórmula concluindo assim que é de facto possível que os aventureiros efetuem a travessia em menos de 17 minutos de acordo como o modelo efetuado.

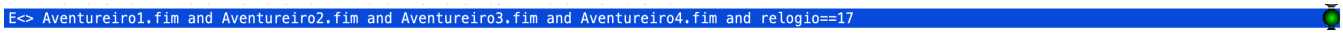

Figura 4.1: Satisfazibilidade 1

- **E<> not (Aventureiro1.fim and Aventureiro2.fim and Aventureiro3.fim and Aventureiro4.fim and relogio<17)**

  Esta propriedade codifica em CTL a impossibilidade de todos os aventureiros atravessarem de um lado para o outro em menos de 17 minutos, tal como foi pedido para verificar,obtive-se a satisfazibilidade da fórmula confirmando que não é possível efetuar a travessia em menos de 17 minutos.

Figura 4.2: Satisfazibilidade 2

5. Conclusão

5.1 Desfecho

Com este trabalho desenvolveu-se a capacidade do grupo abordar a modelação de problemas de lógica temporal e principalmente a familiaridade com a ferramenta *UPAAL*. Como considerações finais achou-se relevante justificar o porquê da junção das acções relacionadas com a lanterna e acções relacionadas com o acompanhamento do viajante que carrega a lanterna no mesmo *template* a que se chamou **lanterna**. Ora isto foi efectuado de maneira a compactar o modelo e simplificar a maneira como se lidou com a modelação da acção de acompanhar o carregador da lanterna, pois separando-a iria-se acrescentar mais complexidade ao modelo. Da maneira modelada controla-se o fluxo de acções efectuadas, ou seja, quando um aventureiro pega na lanterna e está do lado de início vai obrigatoriamente a seguir existir um segundo aventureiro a tomar o lugar de acompanhante.

Garantindo assim umas das premissas para o bom funcionamento do modelo.Enquanto que a separação desta acção do modelo iria obrigar a acrescentar pelo menos mais uma variável de controlo, um *template* e uma guarda na parte do *template* do viajante.