



**UNIVERSIDADE DO MINHO**

Arquitetura e Cálculo

**P1 - LTS and MCRL2**

Rafael Lourenço  
(A86266)

# Conteúdo

<b>1</b>	<b>Resolução dos Exercícios</b>	<b>2</b>
1.1	Exercício 1	2
1.2	Exercício 2	5
1.2.1	Alínea A:	5
1.2.2	Alínea B:	7
1.2.3	Alínea C:	9
1.2.4	Alínea D:	10
1.2.4.1	Safety	10
1.2.4.2	Liveness	10
1.3	Exercício 3	12
1.3.1	Alínea a	12
1.3.2	Alínea b	12

# Capítulo 1

## Resolução dos Exercícios

### 1.1 Exercício 1

O *mCRL2* é uma linguagem que permite a modelação e análise do comportamento de sistemas. Existe também a ferramenta *mRCL2*, que permite obter os **grafos de transição**, verificar a veracidade de **propriedades modais**, também podemos comparar sistemas LTS, nomeadamente saber se dois sistemas são **bissimulares** etc. No entanto, a primeira tarefa é elaborar a especificação do sistema, sendo esta definida em três fases distintas, representadas por 3 *keywords*:

- act - Aqui são definidas todas as ações do sistema. Uma ação consiste numa operação que a máquina ou o utilizador terão de executar para realizar uma operação. Por exemplo, inserir 1 euro numa máquina é uma ação feita pelo utilizador.
- proc - Esta *keyword* serve para definir a sequência de ações feita por um processo (ex: o utilizador), ou seja, aqui tem de se definir todos os possíveis comportamentos para um processo.
- init - Por último, esta *keyword* serve para definir os processos que deverão ser corridos para iniciar o sistema. É possível colocar vários processos a correr concorrentemente através do símbolo  $\parallel$ . No entanto, aqui também são definidas mais 2 funcionalidades:
  - allow - Esta primeira **keyword** serve para colocar todas as funcionalidades que o sistema pode executar, isto é, todas os processos que sincronizam as ações entre si, apenas será colocada a ação que representa a **sincronia** de duas.
  - comm - Nesta secção são definidos todos os processos que são sincronizados.

Dado estas definições, apresento agora um exemplo de uma máquina que permite vender CocaCola's e pepsi's, como é representada na seguinte imagem:



Figura 1.1: Máquina real do sistema

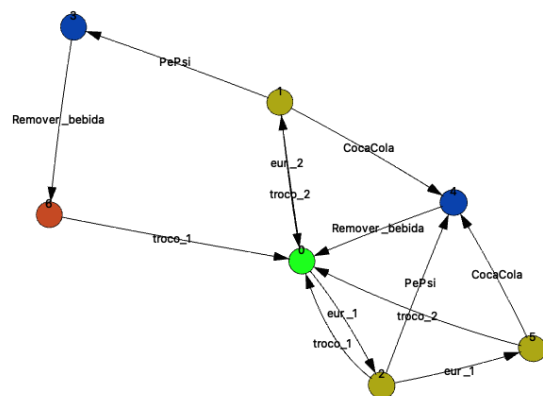


Figura 1.2: Grafo de transição do sistema

Para simplificar as especificações desta máquina, inicialmente apenas serão permitidas moedas de 1 e 2 euros, onde os preços definidos para uma Pepsi é de 1 € e a CocaCola é de 2 €. Sendo assim, este modelo inicial permitirá remover as moedas inseridas antes de fazer um pedido, dar troco de 1 euro no caso da escolha

ser a pepsi e o utilizador tenha colocado 2 €, ou colocação do valor correspondente à bebida sem ter troco. É possível visualizar todos os comportamentos permitidos no grafo de transição, que se encontra acima na figura 1.2. De seguida, é apresentada a especificação necessária para descrever este comportamento.

```
act
  ins1, ins2, acc1, acc2, eur_1, eur_2, chg1, chg2, ret1, ret2, troco_1, troco_2,
  optCo, optPe, putCo, putPe, Remover_bebida, CocaCola, PePsi;
proc
  User = (
    ins1. (optPe + chg1 + ins1.(optCo + optPe.chg1 + chg2 ) )+
    ins2. (chg2 + optPe.chg1 + optCo )
  ).User;
  Mach = (
    acc1.( putPe.Remover_bebida + acc1.( putCo.Remover_bebida + (ret2 + ret1.ret1 ) ) + ret1 ) +
    acc2.( putPe.Remover_bebida.ret1 + putCo.Remover_bebida + (ret2 + ret1.ret1 ) )
  ).Mach ;
init
  allow(
    { eur_1, eur_2, CocaCola, PePsi, troco_1, troco_2, Remover_bebida },
    comm(
      { ins1|acc1 -> eur_1, ins2|acc2 -> eur_2, chg1|ret1 -> troco_1, chg2|ret2 -> troco_2,
        optCo|putCo -> CocaCola, optPe|putPe -> PePsi },
      User || Mach
    ) ) ;
```

Figura 1.3: Especificação total do sistema **LTS** para moedas de 1 e 2 €

Podemos verificar que, o que um utilizador poderá realizar é, inserir 1 € (**ins1.**) e após isso poder fazer 3 ações distintas. Uma delas é escolher a pepsi (**ins1.optPe**), ou remover o dinheiro inserido (**ins1.chg1**), ou inserir mais 1 € e escolher uma CocaCola (**ins1.ins1.optCo**), ou sendo esta a última alternativa, após inserir as 2 moedas de 1 euro, escolher um pepsi e pedir o troco de um euro (**ins1.ins1.optPe.chg1**). Dado isto, o processo para dois euros é similar.

De forma a melhorar o exemplo mostrado anteriormente, criei um segundo exemplo com adição da funcionalidade que permite a inserção de moedas de 50 cêntimos. Com isto, o número de comportamentos do sistema aumenta significativamente, como podemos ver no grafo de transição e na especificação:

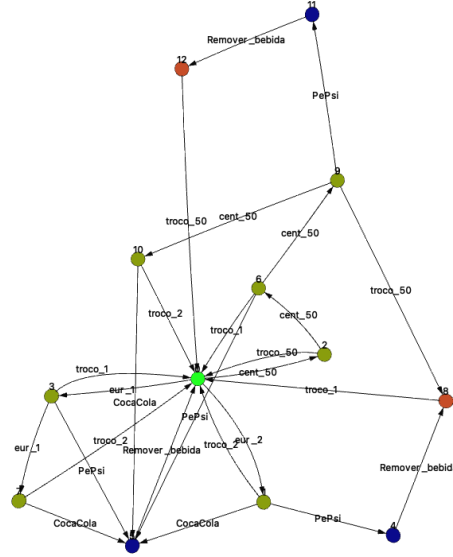


Figura 1.4: Grafo de transição

act

```
ins50, ins1, ins2, acc50, acc1, acc2, cent_50, eur_1, eur_2,
chg50, chg1, chg2, ret50, ret1, ret2, optCo, optPe, putCo, putPe ,
Remover_bebida, CocaCola, PePsi, troco_50, troco_1, troco_2 ;
```

proc

```
User = ( ins50.( chg50 + ins50.(optPe + chg1 + ins50.((chg50.chg1) + (optPe.chg50) +
ins50.(optCo + optPe.chg1 + chg2)))) +
ins1. (optPe + chg1 + ins1.(optCo + optPe.chg1 + chg2 ) )+
ins2. (chg2 + optPe.chg1 + optCo )
).User;
Mach = (
acc50.(ret50 + acc50.( putPe.Remover_bebida + acc50.((ret50.ret1) + (putPe.Remover_bebida.ret1) +
acc50.(putCo.Remover_bebida + ret2)) + ret1 )) +
acc1.( putPe.Remover_bebida + acc1.( putCo.Remover_bebida + (ret2 + ret1.ret1 ) ) + ret1 ) +
acc2.( putPe.Remover_bebida.ret1 + putCo.Remover_bebida + (ret2 + ret1.ret1 ) )
```

).Mach ;

init

```
allow(
{ cent_50, eur_1, eur_2, CocaCola, PePsi, troco_50, troco_1, troco_2, Remover_bebida },
comm(
{ ins1|acc1 -> eur_1, ins2|acc2 -> eur_2, ins50|acc50 -> cent_50,
chg1|ret1 -> troco_1, chg2|ret2 -> troco_2, chg50|ret50 -> troco_50,
optCo|putCo -> CocaCola, optPe|putPe -> PePsi },
User || Mach
) ) ;
```

Figura 1.5: Modelação da maquina com moedas de 50 cêntimos ou 1 e 2 €

Assim foram adicionadas as funcionalidades para fazer qualquer pedido apenas com moedas de 50 cêntimos, ou até mesmo intercaladas com as de 1 e 2 euros. Tiveram de ser adicionadas 5 novas ações, sendo que duas delas são ações síncronas, ou seja, permitem especificar ações que ocorrem em 2 processos distintos.

## 1.2 Exercício 2

### 1.2.1 Alínea A:

Sendo **B** um *buffer* de uma posição e **BS** um *buffer* de duas posições, usando a sincronização de dois B' é possível verificar o seguinte progresso:

$$Bs = (B(in, mo, mi, r) \mid B(mo, out, t, mi)) \setminus \{mo, mi\}$$

$$\equiv \{B - definition\}$$

$$(in \cdot \overline{mo} \cdot mi \cdot \bar{r} \cdot B \mid mo \cdot \overline{out} \cdot t \cdot \overline{mi} \cdot B) \setminus \{mo, mi\}$$

$$\rightarrow \text{Exec. } \mathbf{in}$$

$$(\overline{mo} \cdot mi \cdot \bar{r} \cdot B \mid mo \cdot \overline{out} \cdot t \cdot \overline{mi} \cdot B) \setminus \{mo, mi\}$$

$$\rightarrow \text{Exec. } \mathbf{T\_mo}$$

$$(mi \cdot \bar{r} \cdot B \mid \overline{out} \cdot t \cdot \overline{mi} \cdot B) \setminus \{mo, mi\}$$

$$\rightarrow \text{Exec. } \overline{\mathbf{out}}$$

$$(mi \cdot \bar{r} \cdot B \mid t \cdot \overline{mi} \cdot B) \setminus \{mo, mi\}$$

$$\rightarrow \text{Exec. } \mathbf{t}$$

$$(mi \cdot \bar{r} \cdot B \mid \overline{mi} \cdot B) \setminus \{mo, mi\}$$

$$\rightarrow \text{Exec. } \mathbf{T\_mi}$$

$$(\bar{r} \cdot B \mid B) \setminus \{mo, mi\}$$

$$\rightarrow \text{Exec. } \bar{\mathbf{r}}$$

$$(B \mid B) \setminus \{mo, mi\}$$

$$\rightarrow \text{Chamada recursiva (...)}$$

Dado isto, apresenta-se de seguida um diagrama e o grafo de transição, onde é mostrado este processo através a enumeração dos passos.

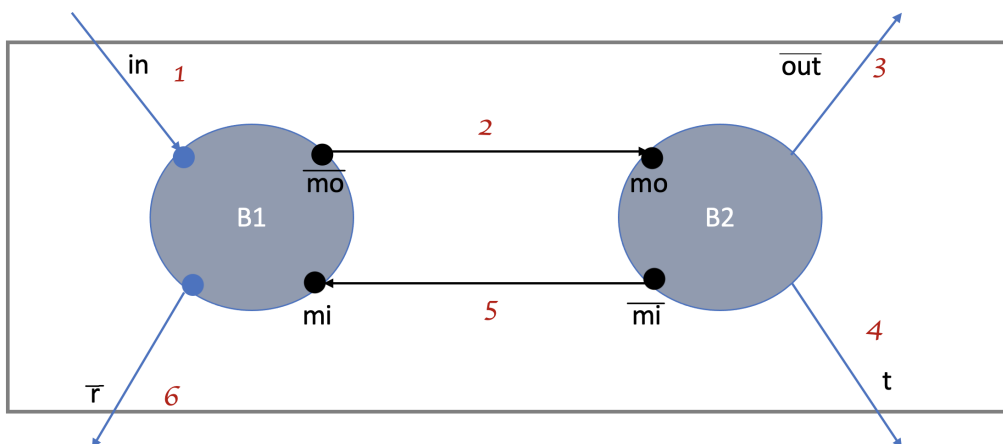


Figura 1.6: Diagrama explicativo

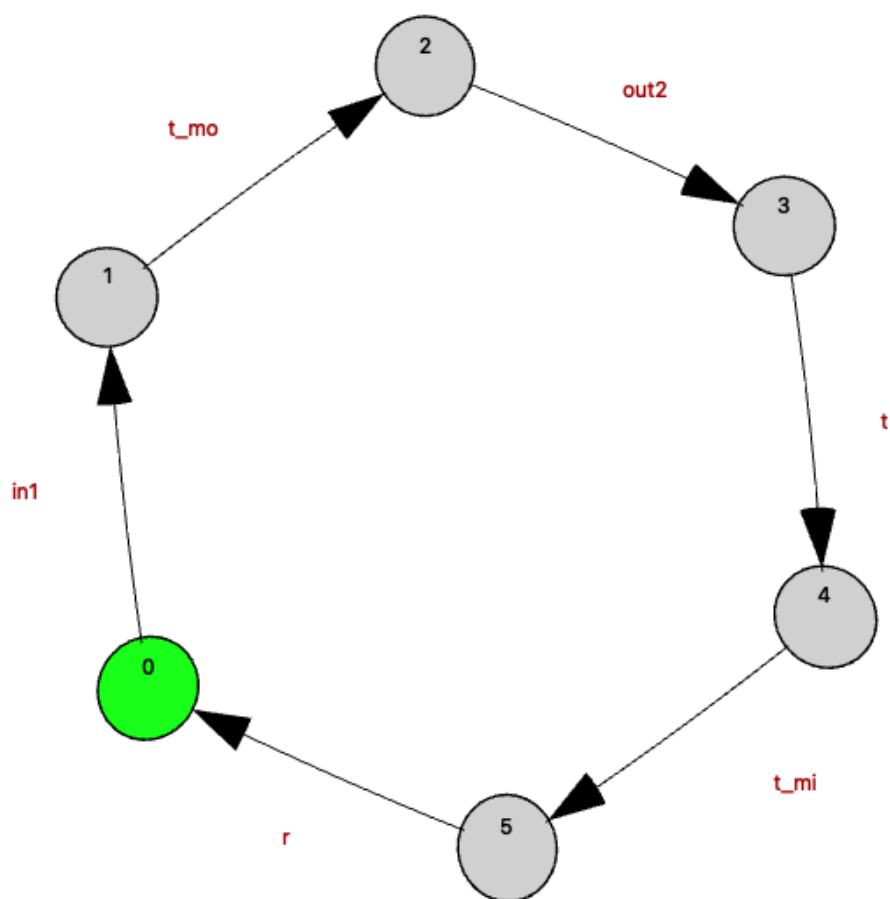


Figura 1.7: Grafo de transição

### 1.2.2 Alínea B:

É possível verificar que nunca é permitido ter as duas posições ocupadas do *buffer*. A solução terá de ter em conta que após ser realizado o **T\_mo (2)** não pode ser executado o **out (3)**. Analogamente, previamente à execução do **out** terá de ser possível executar o **in** novamente. Dado isto, mal seja recebido o input poderemos enviar logo um "ack" a afirmar que já foi recebido esse tal input. Desta forma a definição do *buffer* de uma posição (**B**) passaria a ser a seguinte:

$$B(in, out, t, r) = in \cdot \bar{r} \cdot \overline{out} \cdot t \cdot B$$

Logo, o processo ficaria assim:

$$Bs = (B(in, mo, mi, r) \mid B(mo, out, t, mi)) \setminus \{mo, mi\}$$

$$\equiv \{\mathbf{new} \ B - definition\}$$

$$(in \cdot \bar{r} \cdot \overline{mo} \cdot mi \cdot B \mid mo \cdot \overline{mi} \cdot \overline{out} \cdot t \cdot B) \setminus \{mo, mi\}$$

→ Exec. **in**

$$(\bar{r} \cdot \overline{mo} \cdot mi \cdot B \mid mo \cdot \overline{mi} \cdot \overline{out} \cdot t \cdot B) \setminus \{mo, mi\}$$

→ Exec.  $\bar{r}$

$$(\overline{mo} \cdot mi \cdot B \mid mo \cdot \overline{mi} \cdot \overline{out} \cdot t \cdot B) \setminus \{mo, mi\}$$

→ Exec. **T\_mo**

$$(mi \cdot B \mid \overline{mi} \cdot \overline{out} \cdot t \cdot B) \setminus \{mo, mi\}$$

→ Exec. **T\_mi**

$$(B \mid \overline{out} \cdot t \cdot B) \setminus \{mo, mi\}$$

→ Neste momento já poderia executar o **in** da próxima chamada recursiva e, sendo que ainda não foi executado o **out** verifica-se que é possível colocar dois elementos no *buffer* com esta nova definição. O novo diagrama e grafo de transição ficariam assim:



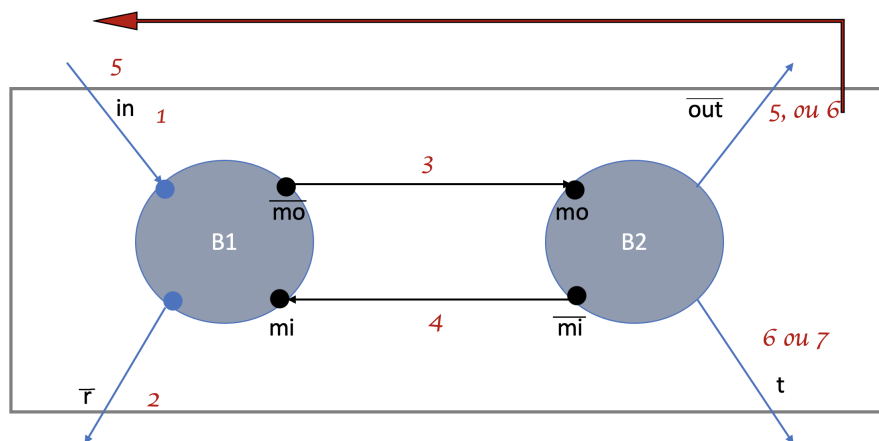


Figura 1.8: Diagrama explicativo

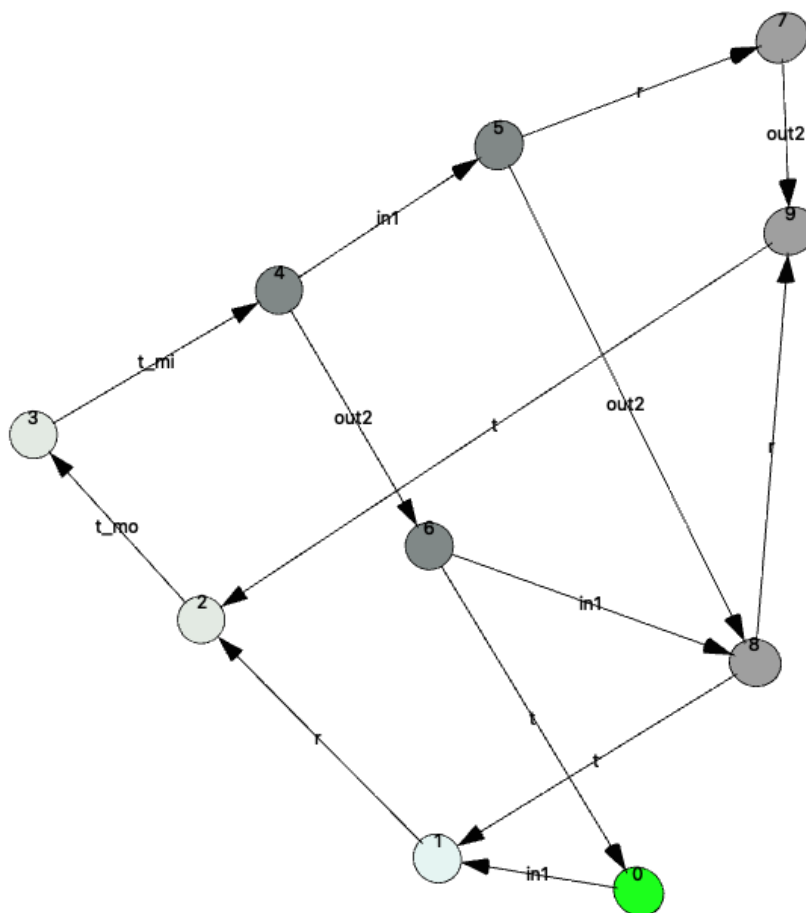


Figura 1.9: Grafo de transição

### 1.2.3 Alínea C:

A nova especificação que mostrei anteriormente permite a realização de *buffers* com tamanho arbitrário. Este tamanho terá de ser pré definido, tal como é dito no enunciado. O que será necessário alterar é o número de ações , processos, ações permitidas no sistema (*allow*), e as ações sincronizadas entre os *buffers* (*comm*). É apresentado de seguida, o exemplo do código para a realização de 2 buffers no MRCL2:

```
act
    in1,mo1,mi1,r,mo2,out2,t,mi2,t_mo,t_mi;

proc
    %Alinea A
    %B1 = in1.mo1.mi1.r .B1;
    %B2 = mo2.out2.t.mi2 .B2;

    B1 = in1.r.mo1.mi1.B1;
    B2 = mo2.mi2.out2.t.B2;

init
    allow(
        {in1,t_mi,t_mo,t,r,out2},
        comm({ mi1|mi2 ->t_mi , mo1|mo2 ->t_mo },
        B1||B2));
```

Figura 1.10: Modelação do *buffer*

Após uma análise detalhada deste código, foi possível deduzir a formula que retornaria o n<sup>o</sup> de ações necessárias, para a realização do mesmo.Dado um *buffer* de tamanho x , a formula seria:

$$\text{ações}(x) = 4+6(x-1)$$

Figura 1.11: Cálculo do número de ações

Para saber o número necessário de processo, é trivial por é proporcional ao número de *buffers* de uma posição, isto é , para um *buffer* de tamanho x irá existir x processos.

$$p(x) = x$$

Figura 1.12: Cálculo do número de processos

Para saber o número necessário de acções permitidas a executar no sistema, bastou deduzir o número ações em sincronia que aumenta à medida que um *buffer* é adicionado, sendo que esse número é 2.No entanto, no caso do primeiro poder fazer um *ack* para o exterior e o último ter de espera por uma *mensagem* temos de adicionar essas duas ações, apresentado a seguinte formula:

$$\text{allow}(x) = 2x+2 = 2(x+1)$$

Figura 1.13: Cálculo do número de ações permitidas

De forma similar, o número de ações sincronizadas é 2 por *buffer*, porém, dado que o primeiro e último *buffer* apenas estão em sincronia com um *buffer*, é necessário retirar 2 ações.

$$\text{comm}(x) = 2x-2 = 2(x-1)$$

Figura 1.14: Cálculo do número de ações sincronizadas

Dado isto, este processo seria fácil de **automatizar** através de uma linguagem de programação que permitisse a escrita em ficheiros, como por exemplo o *python* .

#### 1.2.4 Alínea D:

Apresento de seguida, as propriedades criadas de *safety* e *liveness* que foram testadas no *MCRL2*.

##### 1.2.4.1 Safety

Estas propriedades servem para garantir que o sistema não cometa erros que possam comprometer o sistema.

Esta propriedade de evita que ocorram deadlocks, ou seja, existe sempre uma ação possível de executar.

```
%No deadlock
[true*]<true> true
```

Figura 1.15: Safety propriety

Esta propriedade obriga a que sempre que entra um elemento num *buffer* (B) este terá de sair.

```
%Always (whenever an element enters it leaves)
[true*.in1.!t_mo+.t_mo] true
```

Figura 1.16: Safety propriety

Esta propriedade obriga a que a primeira ação do sistema seja um *in*.

```
%Intial action is in.
[in1.true*]true
```

Figura 1.17: Safety propriety

##### 1.2.4.2 Liveness

Estas propriedades servem para forçar o sistema a evoluir, ou seja, a executar alguma ação.

Esta propriedade apresentada abaixo, obriga a sempre que exista uma ação **in**, o buffer(B) eventualmente irá fazer um **out** (T\_mo).

```
%After in, eventually t_mo.
[in1]<true*.t_mo> true
```

Figura 1.18: Liveness propriety

Esta propriedade faz com que qualquer elemento que entre no sistema eventualmente irá sair.

```
%sempre que entra no sistema sai do sistema!
[in1]<true*.t> true
```

Figura 1.19: Liveness propriety

Por último, esta propriedade obriga que sempre que um **r** seja executado, eventualmente existiu um **in** .

```
<true*.in1>[r] true
```

Figura 1.20: Other propriety

Para mostrar a veracidade das propriedades, esta foram executadas no *mCRL2* e o resultado foi o seguinte.










liveness1	W		
S_DeadLock	W		
S_elemlnf	W		
S_INincio	W		
Live_EnvioACK	W		
Live_SQES	W		

Figura 1.21: Resultado do mCRL2

É de realçar que a negação das propriedades dão todas falso.

## 1.3 Exercício 3

### 1.3.1 Alínea a

Dado dois sistemas  $(S1, N, \rightarrow_1)$  e  $(S2, N, \rightarrow_2)$ , para provar a bissimilaridade entre estes é necessário validar as seguintes duas formulas para todos os pares  $\forall(p, q) \in R$ , sendo  $R$  a relação de bissimilaridade.

$$\begin{aligned} p \xrightarrow{a}_1 p' &\Rightarrow \langle \exists q' : q' \in S2 : q \xrightarrow{a}_2 q' \wedge (p', q') \in R \rangle \\ q \xrightarrow{a}_2 q' &\Rightarrow \langle \exists p' : p' \in S2 : p \xrightarrow{a}_1 p' \wedge (p', q') \in R \rangle \end{aligned}$$

De forma a, mostrar que o sistema **s** e o **t** não são bissimilares apresento o seguinte contra-exemplo para o par  $(s2, t1)$ .

$$s2 \xrightarrow{a}_1 s \Rightarrow \langle \nexists q' : q' \in S2 : t1 \xrightarrow{a}_2 q' \wedge (s, q') \notin R \rangle$$

Similarmente, para os sistemas **s** e **v** temos o seguinte contra-exemplo para o par  $(s2, v3)$ .

$$s2 \xrightarrow{a}_1 s \Rightarrow \langle \nexists q' : q' \in S2 : v3 \xrightarrow{a}_2 q' \wedge (s, q') \notin R \rangle$$

Por último, temos os sistemas **v** e **t** que também não são bissimilares devido ao seguinte contra-exemplo para o par  $(t2, v3)$ .

$$t2 \xrightarrow{a}_1 t \Rightarrow \langle \nexists q' : q' \in S2 : v3 \xrightarrow{a}_2 q' \wedge (t, q') \notin R \rangle$$

Dado isto, prova-se que estes 3 sistemas não são bisimilares entre si. De seguida, passo a mostrar as propriedades modais que **distinguem** estes sistemas.

Para o par de sistemas  $(s, t)$ , a propriedade à esquerda apenas é válida no sistema **s**, e à direita apenas é válida em **t**.

$[a.b] \langle a \rangle \text{true}$

$[a.b] \langle a \rangle \text{false}$

Para o par de sistemas  $(s, v)$ , a propriedade à esquerda apenas é válida no sistema **s**, e à direita apenas é válida em **v**.

$[a.b] \langle a \rangle \text{true}$

**São** simulares.

Para o par de sistemas  $(v, t)$ , a propriedade à esquerda apenas é válida no sistema **v**, e à direita apenas é válida em **t**.

$[a.b.b] \langle b \rangle \text{true}$

$[a.b.b] \langle a \rangle \text{false}$

### 1.3.2 Alínea b

De forma a, mostrar a veracidade da alínea anterior apresento as seguintes imagens:

```

Detected mCRL2 extension.
Detected mCRL2 extension.
Starting to load an Its from the file GrafoS.Its.
Starting to load an Its from the file GrafoT.Its.
comparing LTSs using bisim...
Strictly  $O(m \log n)$  bisimulation partitioner created for 9 states and 8 transitions
The reduced LTS contains at least 2 states and 1 transition.Estimated 0% done.
The current partition contains 1 bunch (of which 1 is nontrivial),and 2 action-block-slices.
The reduced LTS contains 7 states and 8 transitions.Estimated 100% done.
The current partition contains 7 bunches (of which 0 are nontrivial),and 7 action-block-slices.
LTSs are not equal (strong bisimilarity using the  $O(m \log n)$  algorithm [Jansen/Groote/Keiren/Wijs 2019])
false

```

```

Detected mCRL2 extension.
Detected mCRL2 extension.
Starting to load an Its from the file GrafoS.Its.
Starting to load an Its from the file GrafoV.Its.
comparing LTSs using bisim...
Strictly  $O(m \log n)$  bisimulation partitioner created for 11 states and 10 transitions
The reduced LTS contains at least 2 states and 1 transition.Estimated 0% done.
The current partition contains 1 bunch (of which 1 is nontrivial),and 2 action-block-slices.
The reduced LTS contains 8 states and 10 transitions.Estimated 100% done.
The current partition contains 7 bunches (of which 0 are nontrivial),and 7 action-block-slices.
LTSs are not equal (strong bisimilarity using the  $O(m \log n)$  algorithm [Jansen/Groote/Keiren/Wijs 2019])
false

```

```

Detected mCRL2 extension.
Detected mCRL2 extension.
Starting to load an Its from the file GrafoV.Its.
Starting to load an Its from the file GrafoT.Its.
comparing LTSs using bisim...
Strictly  $O(m \log n)$  bisimulation partitioner created for 10 states and 10 transitions
The reduced LTS contains at least 2 states and 1 transition.Estimated 0% done.
The current partition contains 1 bunch (of which 1 is nontrivial),and 2 action-block-slices.
The reduced LTS contains 8 states and 10 transitions.Estimated 100% done.
The current partition contains 8 bunches (of which 0 are nontrivial),and 8 action-block-slices.
LTSs are not equal (strong bisimilarity using the  $O(m \log n)$  algorithm [Jansen/Groote/Keiren/Wijs 2019])
false

```

Figura 1.22: Bisimilaridade entre os 3 sistemas *LTS* no *mCRL2*