Universidade do Minho

mCRL2

Mestrado Integrado em Engenharia Informática

Software Architecture and Calculi

2ºSemestre 2020/21

Aluno:
Etienne Costa

Docente:
Luís Soares Barbosa

4 de abril de 2021

Conteúdo

1 mCRL2 **2**
1.1 Jogo do Galo 4

2 Two-Position Buffer With Acknowledgements **7**
2.1 The Behaviour Of Bs 7
2.1.1 The Solution To The Problem Detected 8
2.2 buffers with an arbitrary, but fixed number of positions 10
2.3 Safety and Liveness Properties 10

3 Labelled Transition Systems **11**
3.1 S,T and V are not bisimilar 11
3.1.1 T $\not\sim S$ 11
3.1.2 S $\not\sim V$ 11
3.1.3 T $\not\sim V$ 12
3.1.4 Propriedades Modais 12
3.2 MCRL2 to verify that they are not bisimilar 14
3.2.1 T $\not\sim S$ 15
3.2.2 S $\not\sim V$ 16
3.2.3 T $\not\sim V$ 16

1 mCRL2

A mCRL2 é uma linguagem de especificação que pode ser usada para analisar, modelar e especificar o comportamento de sistemas distribuídos e protocolos. É baseado numa álgebra de processos, concretamente *Algebra of Communicating Processes*, enriquecendo os processos com certos tipos de dados no ponto de vista da modelação, podendo ainda acrescentar restrições temporais aos mesmos.

Tal como em todas as álgebras de processos, um conceito fundamental existente na linguagem mCRL2 é o de **processo**. **Processos** são definidos à custa da combinação de acções/etiquetas ou mesmo através da composição de outros processos tirando partido de operadores álgebricos.

De seguida é apresentado um pequeno exemplo da modelação de um sistema:

```
act set,alarm,reset;

proc P = set . R ;
     R = reset . P + alarm . R ;

init P ;
```

Este pequeno exemplo representa a modelação de um *despertador*, tendo como conjunto de acções :

- Set : Permite especificar um alarme.
- Reset : Permite fazer reset ao alarme definido.
- Alarm : Simboliza a reprodução do som.

O conjunto destas acções especifica a transição entre os estados do sistema .
De seguida são definidos dois processos, sendo o processo P definido à custa da composição sequencial (.) de uma acção e um outro processo R. O processo R por sua vez é definido pela alternativa (+) da composição sequencial de uma acção e um processo .

O **init** por sua vez especifica o estado inicial do sistema.
Cada processo tem um espaço de estado correspondente ou Labelled Transition System(LTS) que contém todos os estados que o processo pode atingir, junto com as possíveis transições entre esses estados.

De seguida é feita a representação do LTS resultante do sistema modelado acima, podendo verificar-se todos os estados possíveis que o processo pode atingir, junto com as possíveis transições entre esses estados.

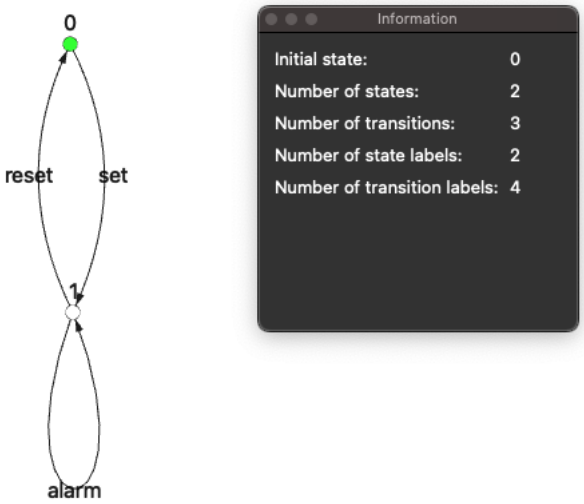

Figura 1: LTS de um despertador .

O processo de execução para gerar um LTS pode ser descrito à custa de 3 fases :

- **Especificação:** A especificação pode ser vista como um modelo de um sistema real, sendo que o seu conteúdo é texto puro na qual representa o modelo na linguagem *mCRL2* podendo ser criado em qualquer editor de texto. Um pequeno exemplo de uma especificação em *mCRL2* é o exemplo do despertador descrito acima, contendo o conjunto de acções e processos.
- **Linearizar:** O processo de linearizar as especificações dá-nos como resultado um *Linear Process Specification(LPS)*, na qual são especificações em *mCRL2* sem qualquer paralelismo associado, prevalecendo o conjunto de condições,acções e regras que especificam como o sistema reage face um certo estímulo. Face a esta simplificação, o LPS é muito mais adequado para análise automatizada. *mcrl22lps é o comando responsável por esta conversão.*
- **Geração do LTS:** Nesta fase podemos gerar o LTS tirando partido do comando *lps2lts*. O resultado produzido por este comando pode ser visualizado pelo comando *ltsgraph* ou *ltsview* que irá tratar da representação do *LTS* resultante .

1.1 Jogo do Galo

Jogo do galo é um jogo de regras extremamente simples, que não traz grandes dificuldades para os seus jogadores e é facilmente aprendido. Consiste em conseguir preencher primeiro uma linha,coluna ou diagonal com um 'X' ou 'O'.

De seguida é feita uma proposta de modelação do jogo à custa de funções.

```
1
2  sort Piece = struct empty | O | X;
3       Board = Pos -> Pos -> Piece;
4
5  map  empty_board:Board;
6       did_win:Piece#Board->Bool;
7       other:Piece->Piece;
8
9  var  b:Board;
10      p:Piece;
11      i,j:Pos;
12 eqn  empty_board(i)(j)=empty;
13      other(O)=X;
14      other(X)=O;
15      did_win(p,b)=(exists i:Pos.(i<=3 && b(i)(1)==p && b(
           i)(2)==p && b(i)(3)==p)) ||
16                   (exists j:Pos.(j<=3 && b(1)(j)==p && b
                        (2)(j)==p && b(3)(j)==p)) ||
17                   (b(1)(1)==p && b(2)(2)==p && b(3)(3)==p
                        ) ||
18                   (b(1)(3)==p && b(2)(2)==p && b(3)(1)==p
                        );
19
20 act  win:Piece;
21      put:Piece#Pos#Pos;
22 proc TicTacToe(board:Board, player:Piece)=
23        did_win(other(player),board)
24           -> win(other(player)).delta
25           <> ( sum i,j:Pos.(i<=3 && j<=3 && board(i)(j)==
                  empty)->
26                   put(player,i,j).
27                   TicTacToe(board[i->board(i)[j->player]],
28                             other(player)));
29
30 init TicTacToe(empty_board,X);
```

Recorrendo ao *sort* conseguimos definir as peças do nosso tabuleiro,sendo que cada posição do tabuleiro pode estar *empty*, com uma *O* ou um *X*.
Recorrendo ao *map* é feita a declaração de três funções *empty_board*, *did_win* e *other*.
Recorrendo ao *eqn* é feita a implementação das mesmas funções:

- **empty_board:** Responsável por colocar todas as posições do tabuleiro vazia.Tendo como parâmetro um tabuleiro, e este por sua vez é declarado como duas posições que mapeiam uma peça, neste caso particular a 'vazia' para todas as posições.
- **did_win:** Responsável por verificar se alguém já venceu o jogo, tirando partido do paralelismo, verifica se já tem alguma linha,coluna ou diagonal preenchida com uma determinada peça.Tem como parâmetro uma peça e o tabuleiro do jogo.
- **other:** Função auxiliar utilizada para ir alternando entre as peças do tabuleiro.

Foram definidas duas *acções* parametrizadas :

- **win:** Responsável por identificar o vencedor do jogo.
- **put:** Responsável por preencher uma determinada posição no tabuleiro com uma determinada peça.

Por fim é especificado o *processo TicTacToe* parametrizado, tendo como parâmetro o tabuleiro do jogo e uma peça/jogador. Antes de qualquer passo é verificado se já houve a vitória do oponente, caso tenha ocorrido, é feita a transição para um novo estado indicando o vencedor do jogo e com o recurso ao *delta* entra-se em deadlock , dando por terminado o jogo. Caso não se verifique a vitória do adversário são dispostas as posições livres do tabuleiro e de seguida executando a *acção* put e feita a chamada recursiva do *processo TicTacToe* com o novo estado do tabuleiro e o outro jogador.

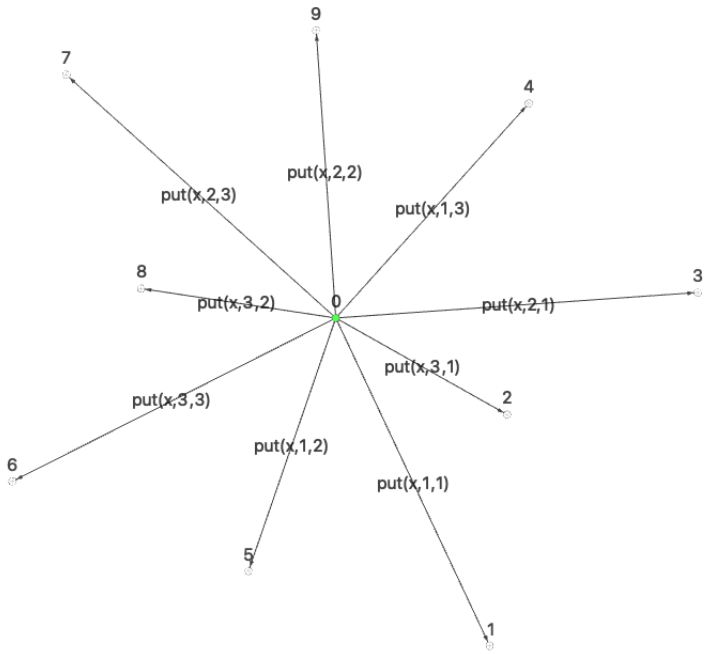

Figura 2: Estado inicial do jogo.

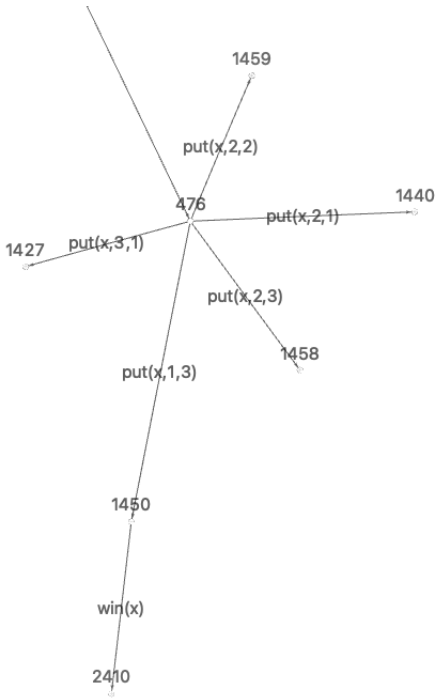

Figura 3: Estado final do jogo, 'X' como vencedor .

2 Two-Position Buffer With Acknowledgements

Consider the following description of a two-position buffer with acknowledgements.Note the process is built from copies of a 1-position buffer also with acknowledgements: it acknowledges in $\bar{r}$ the reception of a message and waits in t the confirmation that a message sent was arrived to its destination.

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

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

2.1 The Behaviour Of Bs

Partindo da seguinte especificação em mCRL2 :

```
act inp,mo,mo_s,mi,mi_s,r_s,out_s,t,m_out,m_r;

proc B1 = inp . mo_s. mi. r_s. B1 ;
proc B2 = mo. out_s . t. mi_s. B2 ;
%proc Bs = allow ( {inp,r_s,out_s,t,m_out,m_r},
%           comm ({mo|mo_s -> m_out, mi|mi_s -> m_r},
%        B1 || B2));


init hide({m_r,m_out},
  allow({inp,r_s,out_s,t,m_out,m_r},
  comm ({mo|mo_s -> m_out, mi|mi_s -> m_r},
  B1 || B2)));
```

Listing 1: Especificação errada em mCRL2 do buffer.

Obtem-se o seguinte LTS :

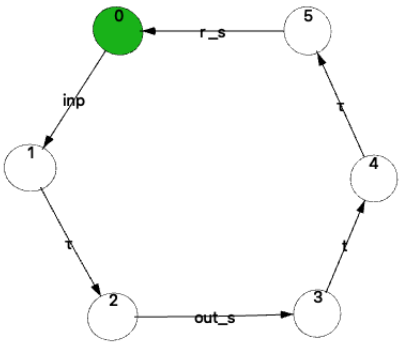

Figura 4: Comportamento incorreto .

É possível verificar que esta especificação não corresponde ao comportamento desejado devido a ordem de execução. Pois um dos problemas que podemos verificar numa fase inicial é que após a realização de uma transição por *inp* é espectável o sistema emitir uma mensagem capaz de confirmar a recepção do elemento, mas não é o que acontece. Sendo assim podemos confirmar que o sistema não tem o comportamento desejado.

Bs = (B(in,mo,mi,r) | B(mo,out,t,mi))$\backslash_{\{mo,mi\}}$
$\Longrightarrow Def$
Bs = (in.$\overline{\text{mo}}$.mi.$\bar{\text{r}}$.B_1|mo.$\overline{\text{out}}$.t.$\overline{\text{mi}}$.B_2)$\backslash_{\{mo,mi\}}$
$\Longrightarrow in$
($\overline{\text{mo}}$.mi.$\bar{\text{r}}$.B_1|mo.$\overline{\text{out}}$.t.$\overline{\text{mi}}$.B_2)$\backslash_{\{mo,mi\}}$
$\Longrightarrow \tau$
(mi.$\bar{\text{r}}$.B_1|$\overline{\text{out}}$.t.$\overline{\text{mi}}$.B_2)$\backslash_{\{mo,mi\}}$
$\Longrightarrow\overline{\text{out}}$
(mi.$\bar{\text{r}}$.B_1|t.$\overline{\text{mi}}$.B_2)$\backslash_{\{mo,mi\}}$
$\Longrightarrow t$
(mi.$\bar{\text{r}}$.B_1|$\overline{\text{mi}}$.B_2)$\backslash_{\{mo,mi\}}$
$\Longrightarrow \tau$
($\bar{\text{r}}$.B_1|B_2)$\backslash_{\{mo,mi\}}$
$\Longrightarrow\bar{\text{r}}$
(B_1|B_2)$\backslash_{\{mo,mi\}}$

2.1.1 The Solution To The Problem Detected

A solução encontrada para o problema detectado, consiste na nova definição do sistema em causa, modificando a posição da transição que acusa a recepção de um determinado elemento no buffer.

Consider the following description of a two-position buffer with acknowledgements.Note the process is built from copies of a 1-position buffer also with acknowledgements: it acknowledges in $\bar{\text{r}}$ the reception of a message and waits in t the confirmation that a message sent was arrived to its destination.

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

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

Partindo da seguinte especificação em mCRL2 :

```
act inp,mo,mo_s,mi,mi_s,r_s,out_s,t,m_out,m_r;

proc B1 = inp . r_s . mo_s . mi . B1 ;
proc B2 = mo . mi_s . out_s . t . B2 ;

init hide({m_r,m_out},
           allow({inp,r_s,out_s,t,m_out,m_r},
     comm ({mo|mo_s -> m_out, mi|mi_s -> m_r},
          B1 || B2 ))) ;
```

Listing 2: Especificação correta em mCRL2 do buffer.

Obtem-se o seguinte LTS :

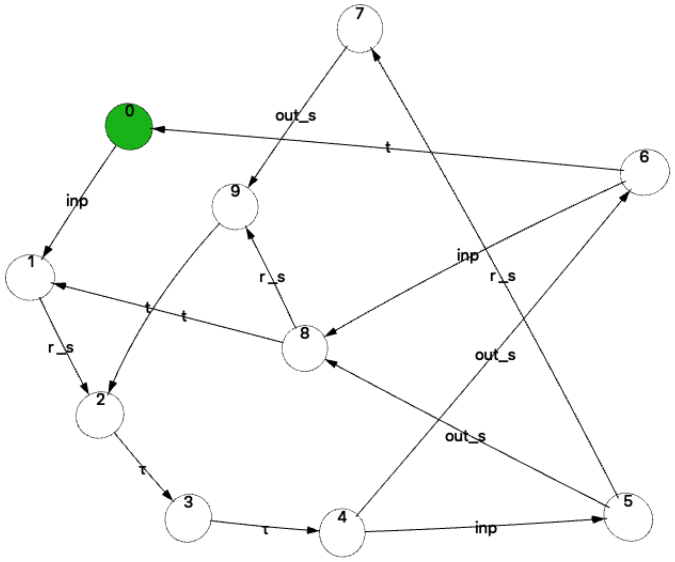

Figura 5: Comportamento correto .

Seguindo este fluxo de transições é possível verificar que com base nesta modificação o sistema passa a ter o comportamento desejado .

Bs = (B(in,mo,mi,r) | B(mo,out,t,mi))$\backslash_{\{mo,mi\}}$
$\Longrightarrow Def$
(in.$\overline{\text{r}}$.$\overline{\text{mo}}$.mi.B_1|$mo.\overline{\text{mi}}.\overline{\text{out}}$.t.$B_2$)$\backslash_{\{mo,mi\}}$
$\Longrightarrow in$
($\overline{\text{r}}$.$\overline{\text{mo}}$.mi.B_1|$mo.\overline{\text{mi}}.\overline{\text{out}}$.t.$B_2$)$\backslash_{\{mo,mi\}}$
$\Longrightarrow\overline{\text{r}}$
($\overline{\text{mo}}$.mi.B_1|$mo.\overline{\text{mi}}.\overline{\text{out}}$.t.$B_2$)$\backslash_{\{mo,mi\}}$
$\Longrightarrow \tau$
(mi.B_1|$\overline{\text{mi}}.\overline{\text{out}}$.t.$B_2$)$\backslash_{\{mo,mi\}}$
$\Longrightarrow \tau$
(B_1|$\overline{\text{out}}$.t.B_2)$\backslash_{\{mo,mi\}}$
$\Longrightarrow\overline{\text{out}}$
(B_1|$t.B_2$)$\backslash_{\{mo,mi\}}$
$\Longrightarrow t$
(B_1|B_2)$\backslash_{\{mo,mi\}}$

2.2 buffers with an arbitrary, but fixed number of positions

É possível generalizar o seguinte sistema de acordo com a seguinte especificação :

```
act inp;
    mo: Int;
    mo_s: Int;
    mi: Int;
    mi_s: Int;
    r_s;
    out_s;
    t;
    m_out: Int;
    m_r;

proc Bi = inp . r_s . mo(0) . mi_s(0) . Bi ;
proc Bm(n: Int) = mo_s(n-1) . mi(n-1) . mo(n) . mi_s(n) . Bm(n) ;
proc Bf = mo_s(2) . mi(2) . out_s . t . Bf;


init allow({inp,r_s,out_s,t,mo|mo_s,mi|mi_s},
     Bi || Bm(1) || Bf);
```

Listing 3: Generalização do sistema.

2.3 Safety and Liveness Properties

```
========= Safety Properties =========

absence of deadlock:

[true*]<true> true
```

Listing 4: Safety Properties.

```
========= Liveness Properties =========

Eventually ins and outs are going to be acknowledged.

[inp]<true*.r_s>true

[out_s]<true*.t>true
```

Listing 5: Liveness Properties.

3 Labelled Transition Systems

Dado dois estados S_1 e S_2, o conjunto de acções N e as respectivas relações binárias $\rightarrow_1$ e $\rightarrow_2$, a relação R $\subseteq$ S$_1$xS_2 é uma bissimulação se R e R° forem uma simulação.

3.1 S,T and V are not bisimilar

Partindo da definição de uma bissimulação mencionada acima temos os seguintes casos:

3.1.1 T $\not\sim S$

T = {(t,s),(t_1,s_1),(t_1,s_2),(t_2,s_2)} *É uma simulação, pois* $T \lesssim S$.

T° = {(s,t),(s_1,t_1),(s_2,t_1),(s_2,t_2)} Não é uma simulação .

T° não é uma simulação , pois existem transições relacionadas ao estado s_2, concretamente as transições por ***'a'*** *e* ***'b'*** *que não estão relacionadas aos estados t_1 e t_2. Com base nisso podemos concluir que* $T \not\sim S$.

A propriedade modal que os distingue é a seguinte :

```
cat modal-property.mcf

[a.b]<a>true

Sendo que é possível fazer essa verificação do seguinte modo :

lts2pbes t.lts -f modal-property.mcf modal-property-t.pbes

lts2pbes s.lts -f modal-property.mcf modal-property-s.pbes

pbes2bool modal-property-s.pbes

true

pbes2bool modal-property-t.pbes

false
```

Listing 6: Propriedade modal.

3.1.2 S $\not\sim V$

S = {(s,v),(s_1,v_1),(s_2,v_2)} É uma simulação, pois $S \lesssim V$.

S° = {(v,s),(v_1,s_1),(v_2,s_1)} Não é uma simulação .

S° não é uma simulação , pois o par $(v_3, s_2) \notin S^\circ$ *.Com base nisso podemos concluir que* $S \not\sim V$.

A propriedade modal que os distingue é a seguinte :

```
cat modal - property.mcf

[a.b]<a>true

Sendo que é possível fazer essa verificação do seguinte modo :

lts2pbes v.lts -f modal-property.mcf modal-property-v.pbes

lts2pbes s.lts -f modal-property.mcf modal-property-s.pbes

pbes2bool modal-property-s.pbes

true

pbes2bool modal-property-v.pbes

false
```

Listing 7: Propriedade modal.

3.1.3 T $\not\sim$ V

T = {(t,v),(t_1,v_1),(t_1,v_2),(t_2,v_2)} É uma simulação, pois $T \lesssim V$.

T° = {(v,t),(v_1,t_1),(v_2,t_1),(v_2,t_2)} Não é uma simulação .

T° não é uma simulação , pois o par $(v_3, t_1) \notin T°$ *.Com base nisso podemos concluir que* $T \not\sim V$.

A propriedade modal que os distingue é a seguinte :

```
cat modal - property.mcf

[a.b]<b>true

Sendo que é possível fazer essa verificação do seguinte modo :

lts2pbes t.lts -f modal-property.mcf modal-property-t.pbes

lts2pbes v.lts -f modal-property.mcf modal-property-v.pbes

pbes2bool modal-property-v.pbes

true

pbes2bool modal-property-t.pbes

false
```

Listing 8: Propriedade modal.

3.1.4 Propriedades Modais

```
[a.b]<a>true s
```

Listing 9: Propriedade modal associada ao estado S.

```
<a.b+>[!a]false
```

Listing 10: Propriedade modal associada ao estado T.

```
[a.b.b+]<a>true && <a.b>[!b]false
```

Listing 11: Propriedade modal associada ao estado V.

3.2 MCRL2 to verify that they are not bisimilar

Numa fase inicial foi feita a conversão dos respectivos labelled transitions systems para uma especificação em mCRL2 :

```
act a,b;
proc P = a . b . S . P ;
proc S = b . S + a ;
init P ;
```

Listing 12: Labelled Transitiom System S.

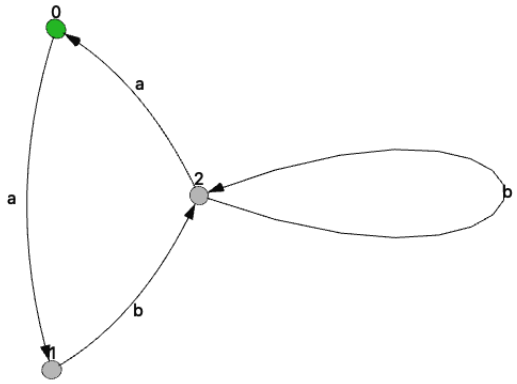

Figura 6: Labelled Transitiom Systems S.

```
act a,b;
proc P = a . S . P ;
proc S = b . S + b . a ;
init P ;
```

Listing 13: Labelled Transitiom System T.

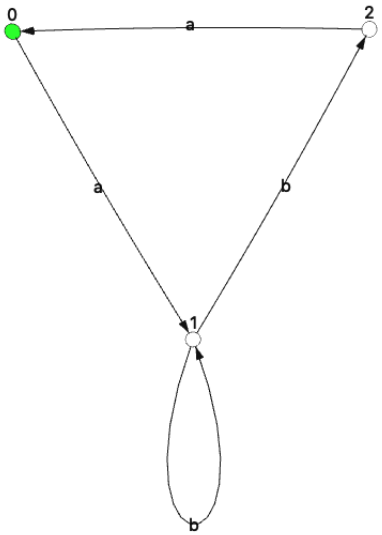

Figura 7: Labelled Transitiom Systems T.

```
act a,b;
proc P = a . ( b . b + b ) . S ;
proc S = b . S + a . P ;
init P ;
```

Listing 14: Labelled Transitiom System V.

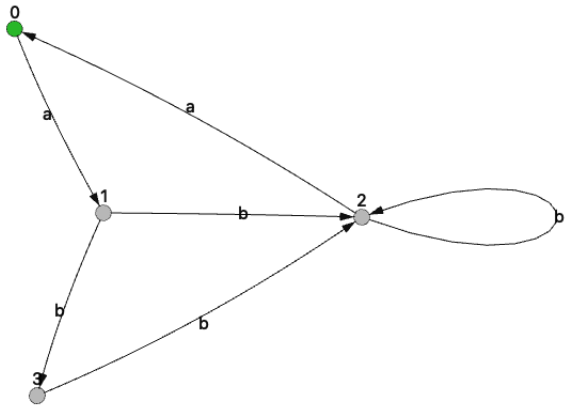

Figura 8: Labelled Transitiom Systems V.

3.2.1 T $\not\sim$ S

```
import os
os.system('mcrl22lps t.mcrl2 t.lps')
os.system('lps2lts   t.lps t.lts')
os.system('mcrl22lps s.mcrl2 s.lps')
os.system('lps2lts   s.lps s.lts')
os.system('ltscompare -ebisim -c t.lts s.lts')
os.system('open *.trc')
```

Listing 15: Python script to prove : T $\not\sim$ S.

Obtendo como resultado algo já esperado,bem como o conjunto de contra-exemplos:

The default bisimulation comparison algorithm cannot generate counter examples.
Therefore the slower gv algorithm is used instead.
Saved counterexample: Counterexample0.trc
Saved counterexample: Counterexample1.trc
LTSs are not equal
false.

```
=== Counterexample0.trc ===
a
b
a
=== Counterexample1.trc ===
a
b
b
```

Listing 16: Contra-exemplos : T $\not\sim$ S.

3.2.2 S $\not\sim V$

```
import os
os.system('mcrl22lps s.mcrl2 s.lps')
os.system('lps2lts   s.lps s.lts')
os.system('mcrl22lps v.mcrl2 v.lps')
os.system('lps2lts   v.lps v.lts')
os.system('ltscompare -ebisim -c s.lts v.lts')
os.system('open *.trc')
```

Listing 17: Python script to prove : S $\not\sim$ V.

Obtendo como resultado algo já esperado,bem como um contra-exemplo:

The default bisimulation comparison algorithm cannot generate counter examples.
Therefore the slower gv algorithm is used instead.
Saved counterexample: Counterexample0.trc
LTSs are not equal
false.

```
=== Counterexample0.trc ===
a
b
a
```

Listing 18: Contra-exemplo : S $\not\sim$ V.

3.2.3 T $\not\sim V$

```
import os
os.system('mcrl22lps t.mcrl2 t.lps')
os.system('lps2lts   t.lps t.lts')
os.system('mcrl22lps v.mcrl2 v.lps')
os.system('lps2lts   v.lps v.lts')
os.system('ltscompare -ebisim -c t.lts v.lts')
os.system('open *.trc')
```

Listing 19: Python script to prove : T $\not\sim$ V.

Obtendo como resultado algo já esperado,bem como o conjunto de contra-exemplos:

The default bisimulation comparison algorithm cannot generate counter examples.
Therefore the slower gv algorithm is used instead.
Saved counterexample: Counterexample0.trc
Saved counterexample: Counterexample1.trc
LTSs are not equal
false.

```
=== Counterexample0.trc ===
a
b
a
=== Counterexample1.trc ===
a
b
b
```

Listing 20: Contra-exemplos : T $\not\sim$ V.