

3º ano da Licenciatura em Ciências da Computação  
**Interação e Concorrência**

7 de Fevereiro de 2021

# Capítulo 1

## Introdução

Este trabalho pretende avaliar na prática toda a matéria referente a sistemas dinâmicos lecionada na unidade curricular de Interação e Concorrência. Para isso, foi-nos pedido que avaliássemos um processo usando uma ferramenta de *model checking*, o *mcrl2*, testando propriedades. Estas serão definidas usando a Lógica de *Hennessy-Milner* e o  $\mu$ -*Calculus*. Adicionalmente, foi também proposta a especificação de um processo que implemente uma fila de espera, baseada na composição paralela de processos.

# Capítulo 2

## Questões

### 2.1 Questão 1

Nesta primeira questão confrontámo-nos com alguns problemas na implementação da definição alternativa dada para o contador, pois não é linearizável pela ferramenta. Através da ferramenta *lpsXSim*, que permite a execução do processo passo a passo, pudemos ver que a definição inicial do nosso processo estava mal. Como faltavam parêntesis em ambas as condições do processo, estava a ser possível executar *dw* a partir de  $C(0)$  e *up* a partir de  $C(n)$ , para  $n > 0$ . A implementação correta da primeira definição foi definida da seguinte forma em mcrl2:

```
act up, dw, zr;

proc C(n:Int) = (n == 0) -> (up.C(1) + zr.C(0))
                  + (n > 0) -> (up.C(n+1) + dw.C(n-1));

init C(4);
```

Figura 2.1: Definição do processo C em mcrl2

Apresentamos agora um traço do processo. Neste exemplo o processo foi inicializado com o valor 4.

Optámos por implementar também uma versão "desenrolada" da definição alternativa do contador, que funciona para  $n \leq 3$ .

Trace		
#	Action	State Change
0		$n\_C := 4$
1	up	$n\_C := 5$
2	up	$n\_C := 6$
3	dw	$n\_C := 5$
4	dw	$n\_C := 4$
5	dw	$n\_C := 3$
6	dw	$n\_C := 2$
7	dw	$n\_C := 1$
8	dw	$n\_C := 0$

Figura 2.2: Traço do processo C inicializado a 4

```
%definicao alternativa do contador, C^n. É um contador de 3 células apenas.
proc Z = up.C(1) + zr.Z;
proc P(n:Int) = dm.C(n-1) + zm.Z;
proc C(n:Int) = (n == 1) -> (up.C(2) + Z)
               + (n == 2) -> (up.C(3) + dw.P(2))
               + (n == 3) -> dw.C(2);

init C(2);
```

Figura 2.3: Definição alternativa do contador

## 2.2 Questão 2

Pretendemos mostrar que  $C^m = Ct_m$ . A prova passa por demonstrar a veracidade dos seguintes factos:

- 1)  $C^m$  e  $Ct_m$  são fracamente bissimilares.
- 2)  $C^m \xrightarrow{\epsilon} (C^m)' \Rightarrow (Ct_m \xrightarrow{\epsilon} X \xrightarrow{\epsilon} (ct_m)')$  e os processos:  $(C^m)'$  e  $(Ct_m)'$  são fracamente bissimilares.
- 3)  $Ct_m \xrightarrow{\epsilon} (Ct_m)' \Rightarrow (C^m \xrightarrow{\epsilon} X \xrightarrow{\epsilon} (c^m)')$  e os processos:  $(C^m)'$  e  $(Ct_m)'$  são fracamente bissimilares.

Para

Vamos efetuar a prova por passos, começando por provar a veracidade da igualdade quando  $m = 0$ .

Começamos por provar que quando  $m = 0$ , os processos são fracamente bissimilares analisando todas as possibilidades de transições começando no par  $(Ct_0, c^0)$ . Por definição,  $C^0 = Z$ . Assumindo que  $(Ct_0, Z) \in R$ , vamos então analisar se possibilidades de transições a partir de  $(Ct_0, Z)$  também estão em  $R$ .

Se  $Ct_0 \xrightarrow{zr} Ct_0$  então  $Z \xrightarrow{zr} Z$  e  $(Ct_0, Z)$  pertence à relação binária  $R$ . E Se  $Z \xrightarrow{zr} Z$  então  $Ct_0 \xrightarrow{zr} Ct_0$  e  $(Ct_0, Z)$  pertence à relação binária  $R$ .

Se  $Ct_0 \xrightarrow{up} Ct_1$  então  $Z \xrightarrow{up} C \setminus Z$  e  $(Ct_1, C \setminus Z)$  pertence à relação binária  $R$ . E Se  $Z \xrightarrow{up} C \setminus Z$  então  $Ct_0 \xrightarrow{up} Ct_1$  e  $(Ct_1, C \setminus Z)$  pertence à relação binária  $R$ .

Vejamos agora todas as transições possíveis a partir de  $(Ct_1, C \setminus Z)$ : Se

$Ct_1 \xrightarrow{dw} Ct_0$  então  $C \sim Z \xrightarrow{dw} (\bar{d}.C + \bar{z}.Z) \sim Z$  e  $(Ct_0, (\bar{d}.C + \bar{z}.Z) \sim Z)$  pertence a relação binária R. E Se  $C \sim Z \xrightarrow{dw} (\bar{d}.C + \bar{z}.Z) \sim Z$  então  $Ct_1 \xrightarrow{dw} Ct_0$  e  $(Ct_0, (\bar{d}.C + \bar{z}.Z) \sim Z)$  pertence a relação binária R.

Por outro lado de  $(Ct_1, C \sim Z)$ , podemos também, efetuar uma transição por up, e por isso se  $Ct_1 \xrightarrow{up} Ct_2$  então  $C \sim Z \xrightarrow{up} C \sim C \sim Z$  e  $(Ct_2, C \sim C \sim Z)$  pertence a relação binária R. E se  $C \sim Z \xrightarrow{up} C \sim C \sim Z$  então  $Ct_1 \xrightarrow{up} Ct_2$  e  $(Ct_2, C \sim C \sim Z) \sim Z$  pertence a relação binária R.

Agora, de  $(Ct_2, C \sim C \sim Z)$  podemos transitar por dw, obtendo de novo o par  $(Ct_1, C \sim Z)$  (a demonstração encontra-se no enunciado). Por outro lado podemos continuar a incrementar o contador transitando por up para  $(Ct_3, (C \sim C) \sim C \sim Z)$ , e o processo continua recursivamente.

Por fim analisamos todas as transições possíveis de  $(Ct_0, (\bar{d}.C + \bar{z}.Z) \sim Z)$ . A partir daqui há apenas uma hipótese: uma transição invisível que corresponde à sincronização de z com zr. Se  $(\bar{d}.C + \bar{z}.Z) \sim Z \xrightarrow{\tau} Z \sim Z$  então  $Ct_0 \xrightarrow{\tau} Ct_0$  e  $(Ct_0, Z \sim Z)$  pertence a relação binária R. Tal como é demonstrado no enunciado,  $Z \sim Z = Z$ . Então o par  $(Ct_0, Z \sim Z)$  é o analisado inicialmente e por isso todas as transições foram analisadas.

Ficou assim provado que  $Ct_0 == C^0$ , uma vez que  $Ct_0$  e  $C^0$  satisfazem as condições de bissimilaridade fraca e as regras 2 e 3, pois não existe nenhuma transição por  $\tau$  a partir de  $Ct_0$  ou  $C^0$ .

Quando  $m > 0$ , facilmente se verifica que  $Ct_m$  e  $C^m$  satisfazem as regras da bissimilaridade fraca. Vamos começar por dizer que  $(Ct_m, C^m) \in R$ . Vamos procurar as transições possíveis a partir de  $(Ct_m, C^m)$ . Se  $Ct_m \xrightarrow{dw} Ct_{m-1}$  então  $C^m \xrightarrow{dw} (\bar{d}.C + \bar{z}.Z) \sim C.... \sim Z$  e  $(Ct_{m-1}, (\bar{d}.C + \bar{z}.Z) \sim C... \sim Z)$  pertence a relação binária R. E se  $C^m \xrightarrow{dw} (\bar{d}.C + \bar{z}.Z) \sim C.... \sim Z$  então  $Ct_m \xrightarrow{dw} Ct_{m-1}$  e  $(Ct_{m-1}, (\bar{d}.C + \bar{z}.Z) \sim C... \sim Z) \in R$ .

Além disso de,  $(Ct_{m-1}, (\bar{d}.C + \bar{z}.Z) \sim C... \sim Z)$  transita por uma ou mais transições por  $\tau$  para  $(Ct_{m-1}, Z)$ . Pois ocorrem m sincronizações de  $\bar{z}$  com zr.

Analisemos em seguida as transições possíveis a partir de  $(Ct_{m-1}, Z)$ . Se  $Ct_{m-1} \xrightarrow{up} Ct_m$  então  $Z \xrightarrow{up} C \sim Z$  e  $(Ct_m, C \sim Z)$  pertence à relação binária R. Além disso, se  $Z \xrightarrow{up} C \sim Z$  então  $Ct_{m-1} \xrightarrow{up} Ct_m$  e  $(Ct_m, C \sim Z) \in R$ .

Do mesmo modo verificamos as transições possíveis a partir de  $(Ct_m, C \sim Z)$ . Se  $Ct_m \xrightarrow{dw} Ct_{m-1}$  então  $C \sim Z \xrightarrow{dw} Z$ . E se  $C \sim Z \xrightarrow{dw} Z$  então  $Ct_m \xrightarrow{dw} Ct_{m-1}$ . Além disto,  $(Ct_m - 1, Z) \in R$ .

Falta agora verificar as transições por up a partir de  $(Ct_m, C^m)$ . Neste caso, se  $Ct_m \xrightarrow{up} Ct_{m+1}$  então  $C^m \xrightarrow{up} C \sim C...Z$  e  $(Ct_{m+1}, (C \sim C) \sim C... \sim Z) \in R$ . Além disso, se  $C^m \xrightarrow{up} (C \sim C) \sim C... \sim Z$  então  $Ct_m \xrightarrow{up} Ct_{m+1}$  e  $(Ct_{m+1}, (C \sim C) \sim C... \sim Z) \in R$ . Facilmente se verifica que é possível avançar por uma transição por dw, para o estado  $(Ct_m, C^m)$

e, além disso, é possível recursivamente continuar a iterar sobre o contador.

Ficou assim provado que  $Ct_m == C^m$ . Os pontos 2 e 3 são verificados pois não existe nenhuma transição por  $\tau$  a partir de  $Ct_m$  ou  $C^m$ .

No final, conclui-se que para todo o  $m$  maior ou igual a zero:  $Ct_m == C^m$ .

### 2.3 Questão 3

Nesta questão, pretende-se verificar se é possível provar a igualdade dos dois processos utilizando o *mcrl2*. Recorrendo a este programa, não é possível verificar directamente a igualdade de processos. Mas, tal como descrito na questão anterior, a prova é feita em três passos: o primeiro consiste em provar que  $C^m$  e  $Ct_m$  são fracamente bissimilares. E o *mCRL2 IDE* permite verificar essa propriedade. As restantes não se conseguem provar computacionalmente e podemos por isso dizer que é possível efectuar uma prova semi-automática.

Contudo, devido às limitações do mCRL2, que não permite definir processos fora da forma concorrente normal, como é o caso da definição do contador (que apresenta recursividade), não conseguimos obter uma especificação do processo que permita fazer a prova.

## 2.4 Questão 4

Foram definidas algumas propriedades úteis que permitem averiguar a correta definição do processo. Apresentamos de seguida 3 propriedades de animação e 3 propriedades de segurança na lógica de *Hennessy-Milner*. Estas seis propriedades foram definidas usando  $\mu$ -calculus, e verificadas no mCRL2.

### 2.4.1 Propriedades de animação

1- Depois de uma ação por *up*, eventualmente o sistema realizará outro *up*.

$[up] \langle up \rangle \text{ true}$

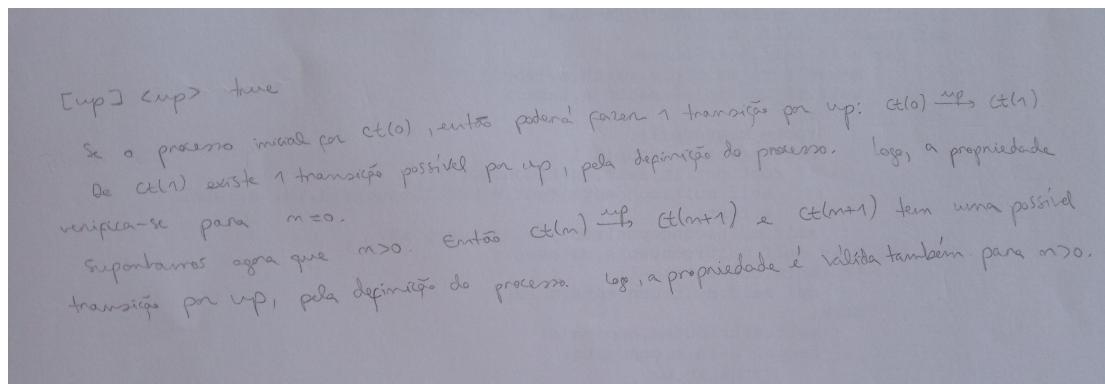


Figura 2.4: Prova da propriedade 1

2- Eventualmente ocorre uma transição por *up* ou por *zr*.

$\langle up \rangle \text{ true} \parallel \langle zr \rangle \text{ true}$

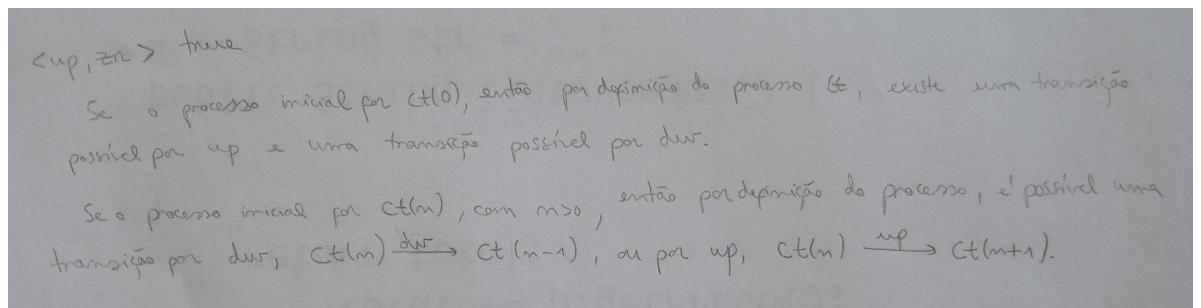


Figura 2.5: Prova da propriedade 2

3- Eventualmente o sistema começará com uma transição por *up*.

$\langle up \rangle \text{ true}$

$\langle up \rangle \text{ true}$   
 Se o processo inicial for  $Ct(0) = 2$ , então é possível realizar  $up$  para  $Ct(1)$ .  
 Se o processo inicial for  $Ct(m)$ , para  $m > 1$ , então é possível realizar 1 ação  
 por  $up$ :  $Ct(m) \xrightarrow{\text{up}} Ct(m+1)$ .

Figura 2.6: Prova da propriedade 3

#### 2.4.2 Propriedades de segurança

4- Quando o contador está a zero, não é possível realizar uma ação por  $dw$ .

[ $zr$ ]  $\langle dw \rangle \text{ false}$

A prova desta propriedade é a seguinte:

Se o processo inicial for  $Ct(0)$  então  $Ct(0) \xrightarrow{zr} Ct(0)$  e de  $Ct(0)$ , por definição do processo, não é possível por  $dw$  imediatamente depois.

Quando  $n > 0$ , não existem transições de  $Ct(n)$  por  $zr$ . Logo, a propriedade é vacuosamente verdadeira.

5- Inicialmente, acontece uma ação por  $dw$  ou por  $up$  ou por  $zr$ .

[ $dw$ ]  $\text{true} \parallel [zr] \text{ true} \parallel [up] \text{ true}$

$\langle dw \rangle \text{ false}$   
 Se o processo inicial for  $Ct(0)$ , então  $Ct(0) \xrightarrow{dw} Ct(0)$  e de  $Ct(0)$ , por definição,  
 não é possível realizar 1 ação por  $dw$  imediatamente a seguir.  
 Quando  $m > 0$ , não existem transições de  $Ct(m)$  por  $dw$ . Logo, a propriedade verifica-se.  
 $\langle dw, up, zr \rangle \text{ true}$   
 Se o processo inicial for  $Ct(0)$  então  $\exists F \in \{Ct' | Ct(0) \xrightarrow{\text{up}} F \vee Ct(0) \xrightarrow{\text{zr}} F\}, F \models \text{true}$   
 Basta tomar  $F = Ct(1)$ , por exemplo.  
 Se  $m > 0$ , então também existe um  $F \in \{Ct' | Ct(m) \xrightarrow{\text{up}} F \vee Ct(m) \xrightarrow{\text{zr}} F\}, F \models \text{true}$ .  
 Basta tomar  $F = Ct(m+1)$ , por exemplo.  
 Assim, fica provado que, para todos  $m$ , é possível realizar uma ação por  $dw$  ou por  $zr$  da  
 forma:  
 por  $up$

Figura 2.7: Prova da propriedade 5

## 6- Ausência de deadlock no processo.

[true] < true > true

[true] < true > true

As possibilidades de transições iniciais quando o m é zero são up e dn.  
Se  $ct(0) \xrightarrow{dn} z$  então, de z, é sempre possível realizar uma ação por dn.  
se  $ct(0) \xrightarrow{up} ct(1)$  então, a partir de  $ct(1)$ , é possível realizar uma ação por up ou por dn.  
Vamos admitir que depois de realizada uma ação em  $ct(m)$ , é possível realizar uma outra ação.  
Então de  $ct(m+1)$  é possível realizar também uma ação. Se for por dn, então obtemos  $ct(m)$  e, pela hipótese de indução,  $ct(m)$  verifica a propriedade. se for por up, obtemos  $ct(m+2)$ , que também permite realizar uma ação.

Figura 2.8: Prova da propriedade 6

Todas estas propriedades validam True no mcrl2.



Figura 2.9: Prova das propriedades no *mCRL2*

As propriedades anteriores podem ser escritas usando  $\mu$ -calculus.

Propriedades de animação:

1)

$$[up] \mu X.(< up > true || < true > X)$$

2)

$$(\mu X.(< up > true || < true > X)) || (\mu X.(< zr > true || < true > X))$$

3)

$$\mu X.(< up > true || < true > X)$$

Propriedade de Segurança:

1)

$$[zr]\nu X.(< dw > false || < true > X)$$

2)

$$[dw]\nu X. < true > X || [zr]\nu X. < true > X || [up]\nu X. < true > X$$

3)

$$[true]\nu X.(< true > true || < true > X)$$

As propriedades anteriores, foram validadas pelo mCRL2, como se mostra na figura seguinte.



Figura 2.10: Prova das propriedades no *mcrl2*

## 2.5 Questão 5

Este exercício para nós foi o mais difícil. Exigiu mais raciocínio para entender bem o que era pedido e como implementá-lo. Inicialmente estávamos a fazer o uso incorreto das portas. Posteriormente, chegámos a esta definição:  $Z$  é a fila de espera vazia, que pode ter uma ação de *enqueue*,  $e$ , e uma ação *zr* para testar se a fila é vazia. Cada célula  $C$  possui uma ação de *enqueue*,  $e$ , e outra de *dequeue*,  $o$ .

```
%quando a porta tem um m no nome significa que é porta de output.
proc Ze = e.(Ce || hide({e,o}, Ze)) + zr.Ze;
proc Pe = om.Ce +zrm.Z;
proc Ce = e.(Ce || hide({e,o}, Ce)) + o.Pe;

init Ce;
```

Figura 2.11: Definição da fila de espera

Vejamos o exemplo de algumas transições típicas.

Pensámos que este processo idealmente deveria ser parametrizado, ou seja, permitir ações, como *enqueue*( $\beta$ ). No entanto, não chegámos a conseguir implementar esta definição.

Como esta versão não é linearizável no *mcrl2*, apresentamos de seguida uma definição alternativa para que possam ser testadas propriedades sobre o processo.

Definimos agora algumas propriedades em *mcrl2*, tendo em conta a de-

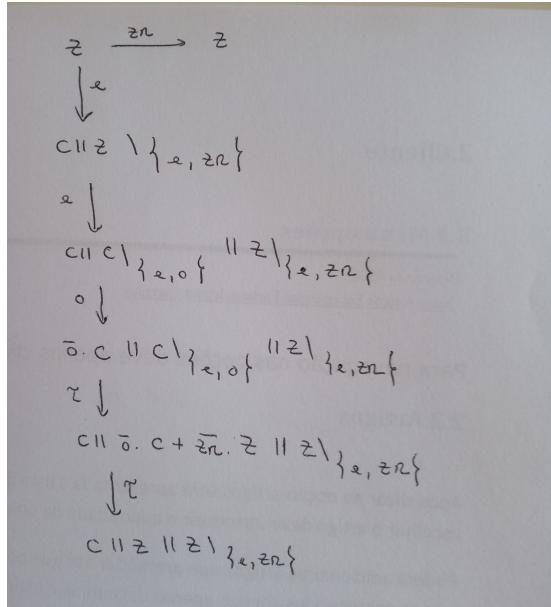


Figura 2.12: Algumas transições do processo

definição alternativa.

### 2.5.1 Propriedades

Definimos duas propriedades de animação:

1. É possível inicialmente realizar um *enqueue*:  $\langle e \rangle \text{ true}$ .  
item Depois de uma ação de *dequeue*, é possível fazer *enqueue*.  
 $[o] \langle e \rangle \text{ true}$

E duas propriedades de segurança:

1. Imediatamente a seguir a verificar se a fila está vazia, não é possível realizar um *dequeue*:  $[zr] \langle o \rangle \text{ false}$
2. Ausência de deadlock:  $[\text{true}*] \langle \text{true} \rangle \text{ true}$

Todas estas propriedades são válidas no *mcrl2*.

```

%definicao alternativa
proc Zel = e.Ce(1) + zr.Zel;
proc Pe(n:Int) = om.Ce(n) +zrm.Z;
proc Ce(n:Int) = (n == 1) -> (e.(Ce(1) + Zel))
+ (n == 2) -> (e.Ce(3))
+ (n == 3) -> (o.Pe(2));
init Ce(2);

```

Figura 2.13: Definição alternativa da fila de espera



Figura 2.14: Propriedades da fila de espera testadas no *mcrl2*

# Capítulo 3

## Conclusão

Este trabalho permitiu estruturar o raciocínio para modelar um problema de processos concreto. Adicionalmente, permitiu explorar com detalhe a ferramenta *mcrl2* e conhecer praticamente todas as suas funcionalidades e limitações. Nomeadamente, fomos confrontadas com a impossibilidade de definir processos recursivos na sua forma mais genérica. A solução passou por encontrar alternativas para a sua implementação, procurando eliminar a recursividade. Esta solução tem a limitação de só funcionar para valores de entrada baixos.

Percebemos também que o principal objetivo deste trabalho consistia na modelação de um problema concreto. A abordagem do problema e o raciocínio no papel foram fundamentais para o poder realizar.