Universidade do Minho

Licenciatura em Ciências da Computação

Interação e Concorrência (3º ano de Curso)
Teste
Resolução

Ivo Lima
(A90214)

15 de abril de 2021

Conteúdo

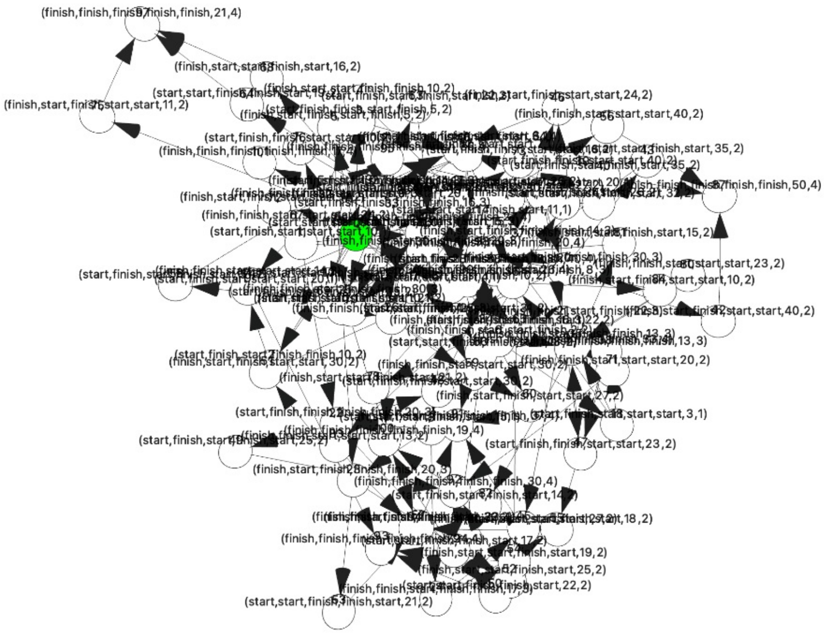
Capítulo 1

Resolução das Questões

1.1 Questão 1

1.

```
sort
    Posicao = struct start | finish;

%---------------------------------------------------------------------------------------------------------------------
act
    avanca_lanterna: Int # Int;
    tras_lanterna: Int;
    avanca_aventureiro: Int # Int;
    tras_aventureiro: Int;
    avanca: Int # Int;
    tras: Int;
    frente_referee: Int # Int;
    tras_referee: Int;
    report: Int;

%---------------------------------------------------------------------------------------------------------------------

proc
    Aventureiro(vel: Int, p:Posicao) =
    (p == start) -> (sum s: Int . (s > vel)-> avanca_aventureiro(vel, s) . Aventureiro(vel,finish)
                                              <>
                                              avanca_aventureiro(s, vel) . Aventureiro(vel,finish))
                                              <>
                                              tras_aventureiro(vel) . Aventureiro(vel,start);

proc
    Flashlight(pos:Posicao) =
    (pos == start) -> sum s,s': Int . avanca_lanterna(s,s') . Flashlight(finish)
                      <>
                      sum s: Int . tras_lanterna(s) . Flashlight(start);

proc
     Referee(minutos:Int,n_acabou:Int) =
     sum s,s': Int . frente_referee(s,s') . Referee(minutos+max(s,s'),n_acabou+2)
     +
     (n_acabou < 4) -> sum s: Int . tras_referee(s) . Referee(minutos + s, n_acabou-1)
                                    <>
                                    report(minutos) . Referee(minutos, n_acabou);

System =
    allow({avanca,tras,report},
        comm({avanca_aventureiro | avanca_lanterna | frente_referee -> avanca, tras_aventureiro | tras_lanterna | tras_referee -> tras},
             Aventureiro(1,start) ||  Aventureiro(2,start) || Aventureiro(5,start) || Aventureiro(10,start)|| Flashlight(start) || Referee(0,0)
    ));

init
      System
;
```



1.2 Questão 2

Para realização desta questão foram utilizados o documento apelidado como *The Formal Specification Language mCRL2*, assim como o link *Tutoriais de mCRL2* que foi disponibilizado pelo coordenador da disciplina.

Tendo em consideração aquilo que está escrito nos documentos acima indicados podemos assumir que o *mCRL2* é uma linguagem de especificação formal, ou seja, deve ser usada durante uma fase de análise de requisitos e, de especificação de programas, descrevendo aquilo que deve ser feito e não como pois as especificações devem sofrer um processo de refinamento antes de serem implementadas de fato, isto é, a adição de detalhes de implementação. Assim como as outras linguagens de especificação é possibilitado a criação de provas matemáticas que validem ou revoguem o software. Para o efeito a mesma possui uma álgebra de processo genérica, baseada em Acp, o cálculo µ completo como uma lógica de especificação assim como um conjunto de ferramentas que lhe permitem modelar, validar e verificar sistemas reativos e também protocolos concorrentes.

Depois deste breve apanhado sobre o que é e qual a funcionalidade da ferramenta *mCRL2*, passaremos então a um exemplo da sua utilização. Para tal finalidade foi escolhido o seguinte caso *Lista telefônica*.

Neste modelo foram impostos os seguintes requisitos: **Armazenar um número de telefone**; **Adicionar e excluir entradas de uma lista telefônica**; **Apresentação de um número dado um nome**.

Atendendo as exigências colocadas podemos identificar as seguintes entidades: **Name**; **PhoneNumber**; **PhoneBook**. Em *mCRL2* pode ser escrito da seguinte maneira:

```
sort Name;
     PhoneNumber;
     PhoneBook = Name -> PhoneNumber;
```

Devemos ter em atenção que um nome pode não ter nenhum número de telefone associado, para lidarmos com esses casos criamos um número especial o *p0*.

```
map  p0: PhoneNumber;
```

De seguida é necessário definir os parâmetros que as ações tomarão e como tal foram definidas as seguintes operações: **addPhone**; **delPhone**; **findPhone**. Deste modo:

```
act  addPhone: Name # PhoneNumber;
     delPhone: Name;
     findPhone: Name;
```

Tomando novamente em consideração a decisão anterior de criar um número especial *p0*, para os nomes sem número associado, podemos ainda especificar que uma lista telefônica vazia mapeia todos os nomes para *p0* numa fase inicial. Tal afirmação será representada do seguinte modo:

```
lambda n: Name . p0;
```

Portanto na modelagem de uma lista telefônica vazia, podemos usar a abstração lambda. Na expressão estamos a definir uma função que recebe argumentos do tipo *Name*, e produz para cada nome um *p0* como resultado pois estes ainda não possuem nenhum número agregado a si. Uma vez que p0 é do tipo *PhoneNumber*,a expressão *lambda n: Name . p0* descreve uma função do tipo *Nome –>PhoneNumber*, que por definição é igual a *PhoneBook*. Dada uma função b do tipo *PhoneBook*, um nome (n) e um número de telefone (p), podemos definir o valor de n em b para p usando a expressão *b [n –>p]*, desta maneira dizemos que todos os nomes *m != n* , *b [n –>p] (m) = b (m)* e *b [n –>p] (n) = p*.

Para além destes casos ainda temos de pensar em como impedir que seja atribuído a *p0* todo e qualquer nome, o que pode ser facilmente evitado salvaguardando a ação *addPhone* com *p! = p0*. Logo após este pequeno problema temos um outro que é ter uma maneira efeciente de encontrar um dado número caso ele exista claro. Para tal existem duas abordagens possíveis: **1.** Supor que o relatório do resultado é imediato e adicionar o número de telefone resultante como um parâmetro para a ação *findPhone*, ou **2.** Supor que a consulta de um número de telefone é assíncrona e, em seguida, dividir a consulta em ação iniciando a consulta (*findPhone*) e uma ação relatando o resultado, por exemplo *reportPhone*. Ambas as abordagens são adequadas pois na primeira situação é retratado um programa síncrono isto é, quando uma tarefa T1 inicia uma segunda tarefa T2, onde é garantido que o T2 seja iniciado e executado dentro do intervalo de tempo de T1 (existente) ou T1 "aguarda"o final de T2 e pode continuar o processamento posteriormente. Nesse sentido, T1 e T2 ocorrem "ao mesmo tempo"(não "em paralelo", mas em um intervalo de tempo contíguo), já na segunda circunstância é retratado um programa assíncrono isto é, o tempo de execução do T2 agora não está relacionado ao T1. Pode ser executado em paralelo, pode ocorrer um segundo, um minuto ou várias horas depois, e o T2 ainda pode ser executado quando o T1 terminar (para processar um resultado do T2, uma nova tarefa T3 pode ser necessária). Nesse sentido, T1 e T2 não estão a ocorrer ao mesmo tempo.

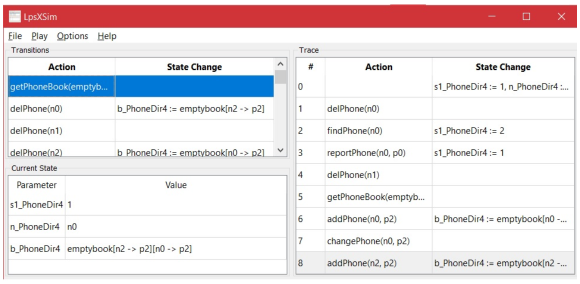
Utilização da 1º abordagem:

```
proc PhoneDir(b: PhoneBook) =
     sum n: Name, p: PhoneNumber . (p != p0) ->
     addPhone(n, p) . PhoneDir(b[n->p])
     + sum n: Name . findPhone(n,b(n)) . PhoneDir()
     + sum n: Name . delPhone(n) . PhoneDir(b[n->p0]);
```

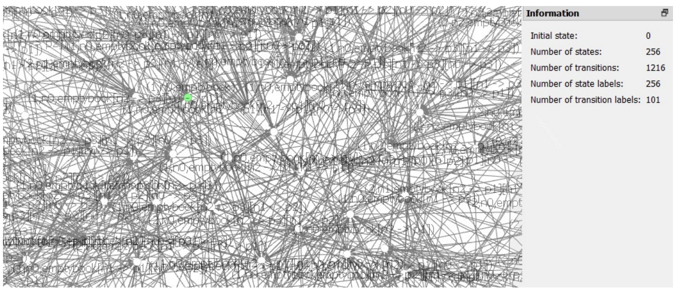
Para a utlização da 2º abordagem, deverá ser acrescentada uma nova ação e a 4º linha de *PhoneDir* deverá ser substituída também.

```
act   reportPhone: Name # PhoneNumber;

proc  PhoneDir(b: PhoneBook) =
      + sum n: Name.findPhone(n).reportPhone(n, b(n)).PhoneDir()
```

Ao verificarmos se o nosso ficheiro está está corretamente formado iremos obter a seguinte notificação *'the file contains a well-formed mCRL2 specification'* e podemos então concluir que temos um especificação formal simples e correta para aquilo foi pedido inicialmente. Caso queiramos assegurar outras propriedades e ter em consideração um maior conjunto de ações, bem como outros problemas que possam aparecer a nossa específicação também irá crescer com ele. Possibilitando a realização de testes como o seguinte: apagar um número, encontrar um número, reportar um número, apagar novamente um número,verificar a lista telefónica, acrescentar um número, mudar um número,...



Podemos pedir à ferramenta um *LTS* que é constituído por um conjunto de estados bem como um conjunto de transições entre esses estados, onde cada uma dessas transições é rotulada por uma ação e um estado é designado como o estado inicial.



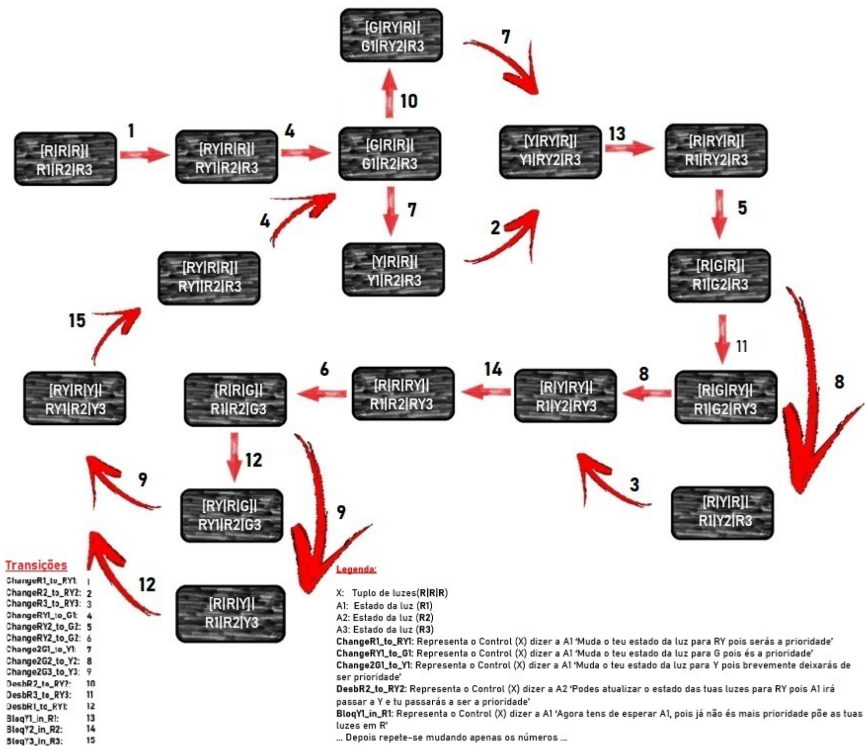
1.3 Questão 3

3. a) assim como a **3. b)**

```
sort
    Cor = struct Green | Red | Red_Yellow | Yellow;

map
    proxCor: Cor -> Cor;

eqn
    proxCor(Red) = Red_Yellow;
    proxCor(Red_Yellow) = Green;
    proxCor(Green) = Yellow;
    proxCor(Yellow) = Red;

%-----------------------------------------------------------------------------------------------------------
act
    change;
    bloqGreen;
    desbloqGreen;
    changeGreen;
    changeYellow;
    desbloq;
    bloq;
    changeRed;
    changeRed_Yellow;
    priori1: Int;
    priori2: Int;
    priori3: Int;

%-----------------------------------------------------------------------------------------------------------

proc

    TrafficLight(st : Cor, s : Int) =
             (proxCor(st) == Green) -> priori2(s).changeGreen.TrafficLight(proxCor(st), s)
             + (proxCor(st) == Red) -> changeRed.TrafficLight(proxCor(st), s)
             + (proxCor(st) == Red_Yellow) -> changeRed_Yellow.TrafficLight(proxCor(st), s)
             + (proxCor(st) == Yellow) -> changeYellow.TrafficLight(proxCor(st), s);


    Control (s : Int) =
              (s<2) -> priori3(s).bloqGreen.desbloqGreen.Control(s + 1)
              +
              (s==2) -> priori3(s).bloqGreen.desbloqGreen.Control(0);


    System =
        allow({change, desbloq, bloq, changeRed, changeRed_Yellow,priori1},
            comm ({bloqGreen | changeGreen -> bloq, desbloqGreen | changeYellow -> desbloq, priori3 | priori2 -> priori1},
                TrafficLight(Red, 0) || TrafficLight(Yellow, 1) || TrafficLight(Red, 2) || Control(0)
        ));


init
     System
;
```

3. c)

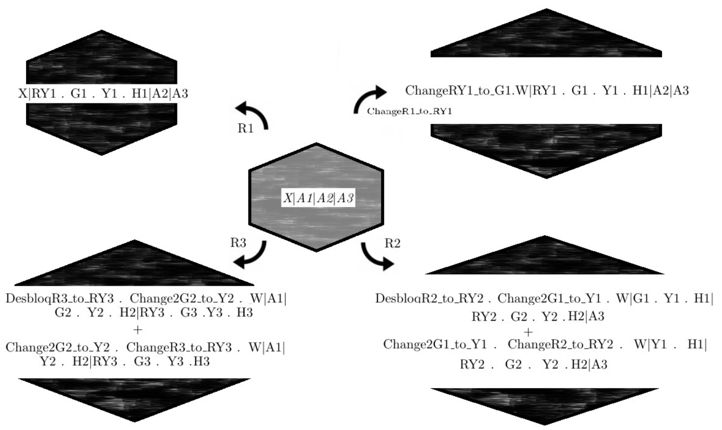
Uma vez que X é o meu *Control*, este terá de considerar e controlar as diversas luzes segundo a prioridade *A1* –>*A2* –>*A3*, tendo isto em mente podemos atribuir a X uma espécie de tuplo onde a alteração desses valores terá como efeito a alteração do estado que as luzes das estradas *A1* *A2* e *A3* apresentam.



3. d)

A1=A2=A3 = R . RY . G . Y . R . RY . G... sendo que neste é um ciclo repetitivo de ações irei omitir essa reescrita atribuindo um H às mesmas. Logo *A1=A2=A3 = R . RY . G . Y . H.*

X terá o basto conjunto de ações enunciadas anteriormente e para não se tornar penosa a escrita de todas elas, estas serão representadas por um W.



(*) *X|A1|A2|A3* $\sim$ *X|RY1 . G1 . Y1 . H1|A2|A3 + ChangeRY1_to_G1.W|RY1 . G1 . Y1 . H1|A2|A3 + ...*

Portanto substitui processos exteriores paralelos por processos exteriores que são somas, sendo estas escolhas não determinísticas, onde os seus argumentos de derivação são * .

Analisando o resultado obtido aplicando o teorema podemos presumir que existe uma grande sincronização entre o processo X e *A1, A2, A3*, ou seja a dependência pedida na pergunta **3. b)** verificasse. Embora existam vários ramos possíveis continua a estar presente um certo ciclo o que pode significar a existência de uma única solução para o problema.

1.4 Questão 4

4. a)

i.

A propriedade descrita nesta alínea é **falsa** e para tal basta apenas apresentar um contraexemplo que prove a falsidade da mesma. Se tomarmos como ponto de partida uma transição *in* em $U1 \triangleright V1$ e tratando-se de uma *bissimulação* o lado de $T \triangleright R$ também deverá conseguir fazer essa mesma transição, mas tal não acontece pois $T \triangleright R$ não têm na sua especificação nenhuma transição rotulada com *in*, o mesmo aconteceria se pensássemos numa transição *out*.

ii.

Neste caso devemos ter em conta a definição de igualdade, que diz o seguinte 'a primeira transição τ deve ser correspondida em ambos os lados', isto é ser possível fazê-la tanto em $U2 \triangleright V2$ como em $U1 \triangleright V1$. Para obtermos esse resultado temos de em ambos os casos recorrer à mudança das variáveis $\overline{out}$ e in por $\bar{c}$ e c respetivamente, feito isso podemos então realizar a transição por τ. Logo podemos então neste momento comprovar que a definição foi respeitada e portanto somos capazes de **aprovar** a propriedade enunciada pelo exercício.

4. b)

• Consideremos

$\mathcal{U} \triangleq (\{c/out\}\mathcal{U}_1 \mid \{c/out\}\mathcal{U}_0) \setminus \{c\}$

$\mathcal{V} \triangleq (\{c/in\}\mathcal{V}_1 \mid \{c/in\}\mathcal{V}_0) \setminus \{c\}$

$\mathcal{U}_1, \mathcal{U}_0, \mathcal{V}_1, \mathcal{V}_0$ estão definidos acima.

$B \triangleq (\mathcal{U} \rhd \mathcal{V}) \rhd T$

$B' \triangleq \mathcal{U} \rhd (\mathcal{V} \rhd T)$

Para provar $B = B'$

$B' = \mathcal{U} \rhd (\mathcal{V} \rhd T)$

$= (\overline{out}.T \mid \overline{out}.\overline{out}.\overline{out}.T) \rhd ((in.R \mid in.in.in.R) \rhd T)$

4.a) ii. $= (\overline{out}.T \mid \overline{out}.T) \rhd ((in.R \mid in.R) \rhd T)$

$= (\overline{out}.T) \rhd ((in.R) \rhd T)$

$B = (\mathcal{U} \rhd \mathcal{V}) \rhd T$

$= ((\overline{out}.T \mid \overline{out}.\overline{out}.\overline{out}.T) \rhd (in.R \mid in.in.in.R)) \rhd T$

4.a) ii. $= ((\overline{out}.T \mid \overline{out}.T) \rhd (in.R \mid in.R)) \rhd T$

4. c)

Olhando para a definição que é apresentada no enunciado do exercício reparamos que U e V têm propriedades distintas, um pode fazer *out's* ou substitui-los por *c's* e o outro *in's* ou substitui-los por *c's*. Mas no contexto deste problema temos um único conjunto o *0*, ou seja toda e qualquer propriedade que *0* tenha será compartilhada consigo mesmo. Logo, ao querermos verificar-se $0 \rhd 0 = 0$ este terá de respeitar a regra anteriormente enunciada de 'a primeira transição τ ser correspondida em ambos os lados' mas como estamos a falar do mesmo conjunto a imposição anterior será trivialmente respeitada e portanto a igualdade será verdade.

1.5 Questão 5

5. a)

Através da análise das expressões reparamos que sempre que o processo $\circlearrowleft nE$ tem capacidade de fazer uma transição por a chegamos ao mesmo estado E mas o seu n é decrementado e quando n toma o valor de 0, E poderá fazer uma nova transição a que o levará a um novo estado *E'*, ou seja neste ultima situação não é feito nada. Em suma podemos considerar o operador $\circlearrowleft$ uma espécie de **replicador**.

5. b)

A expressão apresentada é **verdadeira** quando m toma o valor de 0 e n um valor natural qualquer como por exemplo 3, pois $\circlearrowleft 0E$ não irá alterar em nada E (pois não acrescenta nem reduz o número de transições possíveis) e consequentemente $\circlearrowleft nE$ será trivialmente bissimilar a $\circlearrowleft nE$.

5. c)

Para fazermos a implicação pedida temos inicialmente de considerar o par R={($\circlearrowleft nE$,$\circlearrowleft nF$)|E $\sim$ F}, de seguida creiamos que existe em $\circlearrowleft nE$ uma transição por a para $\circlearrowleft n-1E$ então devido a E e F serem bissimilares tem de existir em $\circlearrowleft nF$ uma transição por a para $\circlearrowleft n-1F$, podemos abreviar todas estas transições por um simples $\sim$. Portanto ficaremos com um R={($\circlearrowleft nE$,$\circlearrowleft nF$)|E $\sim$ F} $\cup \sim$. Porém a demonstração não pode ficar por aqui pois ainda temos os casos de existir em $\circlearrowleft 0E$ uma transição por a para *E'* bem como em $\circlearrowleft 0F$ haver uma transição por a para *F'* mas tal e qual como no exemplo acima apenas temos de adicionar estas transições a R. Em suma precisamos de ter R={($\circlearrowleft nE$,$\circlearrowleft nF$)|E $\sim$ F} $\cup \sim \cup$ {(*E'*,*F'*)|E $\sim$ F} e assim conseguimos demonstrar que a implicação é **verdadeira**.

5. d)

Se fizermos a troca de $\sim$ por $\approx$ tornaria a expressão anterior falsa, uma vez que nesse caso teriamos de analisar as transições *"gordas"*, isto é $\circlearrowleft nE \Rightarrow (a) \circlearrowleft n-1E$ assim 'a' pode ser uma transição simples que nesse caso acontecerá tanto em E como F, o problema surge quando 'a' for uma transição através de τ e dessa maneira acabamos quebrando a equivalência.

$\circlearrowleft nE$: τ.x.0 $\rightarrow(\tau)$ $\tau.\tau$.x.0 $\rightarrow(\tau)$ ...

$\circlearrowleft nF$: x.x.0 $\rightarrow$(x) x.x.x.0 $\rightarrow (x)$ ...

5. e)

Neste caso a solução é óbvia temos de forçar que o primeiro τ seja correspondido por outro τ ou levar a que quando um τ é feito E ficar igual a F, ou o contrário F ficar igual a E. Tendo em conta que $E \neq \tau.E$, mas que $\tau.E = \tau.\tau.E$ podemos alterar a semântica do problema para:

$$\frac{E \xrightarrow{a} E' \quad F \xrightarrow{a} F'}{\circlearrowleft_n \mathsf{E} \xrightarrow{\tau} \circlearrowleft_{n-1} \mathsf{F}} \quad \frac{F \xrightarrow{a} F' \quad E \xrightarrow{a} E'}{\circlearrowleft_n \mathsf{F} \xrightarrow{\tau} \circlearrowleft_{n-1} \mathsf{E}}$$

$$\vee$$

$$\frac{E \xrightarrow{a} E' \quad F \xrightarrow{a} F'}{\circlearrowleft_n \tau.E \xrightarrow{\tau} \circlearrowleft_{n-1} \mathsf{F}} \quad \frac{E \xrightarrow{a} E' \quad F \xrightarrow{a} F'}{\circlearrowleft_n \mathsf{E} \xrightarrow{\tau} \circlearrowleft_{n-1} \tau.F}$$