Lição 4: Estudo de Casos

Luís Soares Barbosa

Sumário

Esta Lição é integralmente dedicada ao estudo de alguns exemplos de modelação e análise de processos com alguma complexidade. Para isso capitaliza na matéria trabalhada nas duas Lições anteriores e faz apelo ao recurso a uma ferramenta de animação de processos — o CWB-NC *— igualmente já introduzida.*

Alguns dos casos discutidos abordam problemas de reconfigurabilidade de redes de processos em tempo de execução que só encontrarão uma formulação genérica no contexto do π-calculus*, cujo estudo se inicia na Lição seguinte. Fazer a ponte entre os dois cálculos que se abordam neste curso é, assim, um segundo objectivo desta Lição.*

1 Especificação *versus* Implementação

Como sabemos, o teorema da expansão permite relacionar processos concorrentes com processos que exibem um comportamento sequencial não-determinístico. É usual tomar o primeiro tipo de processos como realizações *distribuídas* dos do segundo e utilizar a teoria equacional da relação $=$ para estudar essas implementações.

O uso dos termos *especificação* e *implementação* neste contexto pode parecer um pouco forçado na medida em que o *critério de correcção* que se adopta é uma equivalência, tipicamente a congruência observacional $=$. O aumento de *redundância* na passagem da especificação para a implementação não tem pois a ver com o comportamento observável dos processos, que permanece inalterado, mas antes com o grau de paralelismo (que aumenta). Apesar de não explorarmos aqui esse caminho, é relevante salientar que Uma ordem de redundância comportamental em $\mathbb{P}$ pode ser obtida considerando a existência, não de uma bissimulação, mas de uma *simulação* do processo que constitui a especificação no processo implementador.

As implementações típicas na aplicação prática de CCS (*cf.*, por exemplo [Mil89] ou [Fen96]) surgem como variantes da *forma concorrente canónica*

$$\mathsf{new}\, K\ (\{f_1\}\, E_1 \mid \{f_2\}\, E_2 \mid ... \mid \{f_n\}\, E_n)$$

de que já falamos. O leitor é convidado a consultar com alguma atenção o final do capítulo 5 e todo o capítulo 6 de [Mil89], realizando os exercícios propostos, de modo a se tornar confiante no uso das técnicas de equivalência introduzidas. Note-se, em particular, que é possível (e usual) definir bissimulações entre processos que diferem na possibilidade de *divergência*, por exemplo em que um deles (mas não o outro) tem a possibilidade de se comprometer num ciclo perpétuo de acções não observáveis. A verificação da correcção do *Alternating Bit Protocol* (páginas 145 e 149) é particularmente instrutiva a este respeito. Geralmente a convergência de um processo pode ser estabelecida por um argumento autónomo, *e.g.*, utilizando técnicas que analisaremos mais tarde, na Lição 7.

Notas de apoio às aulas de *Interacção e Concorrência*, Lic. em Ciências da Computação, 2014-15.

Particularmente interessante são os casos em que a estrutura concorrente não é fixa de uma vez por todas mas evolutiva. Conseguem-se, assim, exprimir algumas situações em que o sistema se reconfigura dinamicamente. O caso mais simples é o da terminação de um ou mais factores numa composição paralela, que diminui o número de factores na composição. Por outro lado, a invocação recursiva de um processo concorrente num dos seus próprios factores provoca uma "explosão" de paralelismo, eventualmente originando sistemas com conjuntos de estados não finitos. É o caso de $\underline{fix}\ (X = \mathsf{new}\, a\ (... + a.X \mid \overline{a}.X + ...))$. Interessam-nos, sobretudo, versões controladas dessa "explosão" que nos forneçam implementações distribuídas dos sistemas que concebemos.

Neste contexto, vamos discutir algumas dessas implementações, como *case studies* na teoria dos processos que temos vindo a abordar.

Nota 1 **[Cenas dos próximos capítulos]**

Devemos, no entanto, sublinhar que aquilo que se entende por *reconfiguração dinâmica* num sistema distribuido vai além da simples alteração do número de instâncias conectadas. De facto, exige-se que a estrutura de conexão das componentes de um processo seja, ela própria, dinamicamente modificável. O objectivo será permitir, por exemplo, que, num dado instante, a interação que existia entre os sub-comportamentos A e B de um dado processo, através de uma porta a, seja suspensa e, utilizando a mesma porta, A passe a interagir com outro sub-comportamento C.

O conceito subjacente de *mobilidade* exprime esta capacidade de, não apenas as componentes de um processo poderem estar arbitrariamente ligadas, como o facto dessa ligação poder ser modificada como resultado da interacção que estabelece. A *mobilidade* não é directamente exprimível em CCS, nem nas álgebras de processos mais comuns, nem ainda em outros modelos semânticos da concorrência (por exemplo nas Redes de Petri [Pet62]). Tem, contudo, um lugar importante nas abordagens *orientadas ao objecto* embora geralmente com uma formalização insuficiente.

O *π-calculus*[MPW92, Mil99] é uma tentativa de acomodar esta noção numa extensão adequada de CCS. A ideia base é a inclusão de nomes de ligações entre processos como parâmetros das acções, abolindo-se a distinção entre os identificadores de portas (acções) e os dados. Obtém-se, deste modo, uma teoria de primeira ordem para exprimir propriedades de ordem superior.

Analisemos, então, alguns exemplos de raciocínio equacional sobre processos. Os casos a abordar ilustram situações típicas na especificação e implementação de sistemas sobre plataformas computacionais envolvendo distribuição e concorrência.

2 Contadores

Vamos partir da seguinte especificação de um contador, em termos de três acções: up (incremento), dw (decremento) e zr (teste para o valor 0):

$$\begin{aligned} Ct_0 &\triangleq up.Ct_1 + zr.Ct_0 \\ Ct_n &\triangleq up.Ct_{n+1} + dw.Ct_{n-1} \qquad \text{para } n > 0 \end{aligned}$$

Uma possível implementação distribuída deste contador pode ser obtida pela composição paralela de pequenas células — processos do tipo C — cada uma das quais representa uma unidade. As células tem capacidade de se multiplicarem em resposta a uma operação de incremento assim como de se extinguirem na ocorrência de um decremento. Cada uma dispõe de duas portas auxiliares ($\overline{d}$ e $\overline{z}$) usadas para propagação dos decrementos. O número 0 é representado por uma célula especial — o processo Z — que não admite a realização de decrementos, e, portanto, não tem as portas auxiliares referidas. O essencial na dinâmica destes processos é a estrutura

de sincronizações que, para melhorar a legibilidade, se abreviou sob a forma de um operador derivado ($\frown$). Temos, pois,

$$C \triangleq up.(C \frown C) + dw.P$$
$$P \triangleq \overline{d}.C + \overline{z}.Z$$

$$Z \triangleq up.(C \frown Z) + zr.Z$$

onde o operador $\frown$ se define como

$$E \frown F \;=\; \mathsf{new}\; A\; (\{u'/u, z'/z, d'/d\}\; E \mid \{u'/up, z'/zr, d'/dw\}\; F)$$
$$\text{onde}\;\; A = \{u', z', d'\}$$

Note-se que não sendo u uma acção de E, a sua renomeação não tem significado. Por outro lado, no processo F, ao renomearmos up para u' e posteriormente incluirmos u' no conjunto das restrições, estamos a tornar interna a porta up do segundo operando de $\frown$, *i.e.*,, a impedir a sua utilização pelo exterior.

O número de células C activas depende do estado corrente da contagem. Inicialmente o contador pode ser inicializado a 0, caso em que é representado pelo processo $C^0 \triangleq Z$. No caso geral, representamos o estado do contador associado ao número n pelo processo $C^n \triangleq C \frown C \frown C \frown ... \frown Z$, com exactamente n células C. A análise de algumas transições típicas ajudará a perceber a dinâmica do sistema. Note-se que a célula mais externa pode realizar um incremento ao mesmo tempo que um decremento se propaga nas células interiores.

- $C^0 = Z \xrightarrow{up} C \frown Z$.
- $C^1 = C \frown Z \xrightarrow{dw} (\overline{d}.C + \overline{z}.Z) \frown Z$. A partir daqui há apenas uma hipótese: uma transição invisível que corresponde à sincronização de $\overline{z}$ com zr. Temos, então, $(\overline{d}.C + \overline{z}.Z) \frown Z \xrightarrow{\tau} Z \frown Z$. Mas $Z \frown Z = Z$, porque

$$\begin{aligned} Z \frown Z \;&=\; \mathsf{new}\; A\; (\{u'/u, z'/z, d'/d\}\; Z \mid \{u'/up, z'/zr, d'/dw\}\; Z) \\ &=\; \mathsf{new}\; A\; (\{u'/u, z'/z, d'/d\}\; Z) \mid \mathsf{new}\; A\; (\{u'/up, z'/zr, d'/dw\}\; Z) \\ &=\; Z \mid \mathbf{0} \\ &=\; Z \end{aligned}$$

- $C^2 = C \frown C \frown Z \xrightarrow{dw} (\overline{d}.C + \overline{z}.Z) \frown C \frown Z$. Agora a única possível transição é a sincronização de $\overline{d}$ com dw, que origina uma transição por τ para $C \frown (\overline{d}.C + \overline{z}.Z) \frown Z$. De seguida $C \frown (\overline{d}.C + \overline{z}.Z) \frown Z \xrightarrow{\tau} C \frown Z \frown Z$ e, como vimos acima, $C \frown Z \frown Z = C \frown Z = C^1$.

Vamos, agora, verificar a correcção desta implementação relativamente à especificação original. Notemos que C^n pode ser escrito como $C \frown C^{n-1}$. Claramente, $C^0 = up.C^1 + zr.C^0$. Para o caso geral, aplicando o teorema da expansão, obtemos:

$$
\begin{aligned}
C^n &= C \frown C^{n-1} \\
&= up.((C \frown C) \frown C^{n-1}) + dw.(P \frown C^{n-1}) \\
&= up.C^{n+1} + dw.(P \frown C^{n-1}) \\
&= up.C^{n+1} + dw.C^{n-1}
\end{aligned}
$$

Nota 2

O último passo, parecendo intuitivo, carece de verificação. À primeira vista parece que necessitamos de provar que $P \frown C^n = C^n$, igualdade que não é verdadeira. Podemos, contudo, verificar uma propriedade mais fraca, a saber, para todo o $n \geq 0$, $P \frown C^n \approx C^n$.

Prova. Procedamos por indução sobre n. Para $n = 0$, vem

$$
\begin{aligned}
P \frown C^0 &= P \frown Z \\
&= \tau.(Z \frown Z) \\
&= \tau.Z \\
&\approx Z \\
&= C^0
\end{aligned}
$$

Para $n > 0$, suponhamos, como hipótese de indução, que o resultado é válido para $n-1$, *i.e.*,, $P \frown C^{n-1} \approx C^{n-1}$. Então,

$$
\begin{aligned}
P \frown C^n &= \tau.(C \frown (P \frown C^{n-1})) \\
&\approx \tau.(C \frown C^{n-1}) \\
&\approx C \frown C^{n-1}
\end{aligned}
$$

□

Será este resultado suficiente para justificar o último passo da cadeia de equivalências acima? É-o de facto, porque, como sabemos, se $E \approx F$ podemos concluir, para $x \in Act - \{\tau\}$, $x.E = x.F$. No caso em mãos, x é a acção dw como o leitor pode confirmar.

Este exemplo é ilustrativo do papel da equivalência observacional $\approx$ no cálculo. De facto, em muitas situações, boa parte do cálculo equacional pode ser feito em termos de $\approx$, introduzindo-se a congruência $=$ apenas no final.

Temos portanto

$$
\begin{aligned}
C^0 &= up.C^1 + zr.C^0 \\
C^n &= up.C^{n+1} + dw.C^{n-1} \qquad \text{para } n > 0
\end{aligned}
$$

o que permite concluir que tanto a especificação como a implementação são soluções do mesmo sistema de $\mathbb{N}$ equações a $\mathbb{N}$ incógnitas:

$$X^0 = E(X^0, X^1) = up.X^1 + zr.X^0$$
$$X^n = E'(X^{n-1}, X^{n+1}) = up.X^{n+1} + dw.X^{n-1} \qquad \text{para } n > 0$$

De facto, $C^0 = E[f]$ e $Ct_0 = E[g]$, enquanto $C^n = E'[f]$ e $Ct_n = E'[g]$, para $f = [C^i/X^i]_{i\in\mathbb{N}}$ e $g = [Ct_i/X^i]_{i\in\mathbb{N}}$. Logo, para todo o $m \geq 0$, $C^m = Ct_m$ como esperavamos.

3 Stacks & Afins

Um mesmo tipo de implementação distribuída pode ser feito para processos associados a estruturas de dados mais complexas. Vamos analisar aqui o caso das stacks em que os três operadores habituais ($push$, pop e $empty$) actuam sobre uma sequência de elementos de um conjunto qualquer A. Partimos da seguinte especificação:

$$St(\epsilon) \triangleq push(x).St(x) + empty.St(\epsilon)$$
$$St(a:s) \triangleq push(b).St(b:a:s) + \overline{pop}\langle a\rangle.St(s)$$

A implementação que procuramos baseia-se na composição paralela de processos C capazes de armazenarem, cada um, um elemento da stack. Como no caso anterior, cada C dispõe de duas portas adicionais ($\overline{e}$ e o) para propagação de valores ao longo da cadeia (só que agora as duas portas auxiliares têm sentidos de comunicação opostos entre si!). Igualmente se introduz um processo próprio (Z) para representar a stack vazia. Assim,

$$C(x) \triangleq push(y).(C(y) \frown C(x)) + \overline{pop}\langle x\rangle.P$$
$$P \triangleq o(x).C(x) + \overline{e}.Z$$

$$Z \triangleq push(y).(C(y) \frown Z) + empty.Z$$

onde o operador $\frown$ se define agora como

$$E \frown F \;=\; \text{new } A\ \{u'/u, e'/e, o'/o\}\ E \mid \{u'/push, e'/empty, o'/pop\}\ F$$
$$\text{onde } \; A = \{u', e', o'\}$$

Uma stack com a sequência $a_1a_2a_3...a_n$ como valor corrente será representada pela conexão de n células C, cada uma armazenando um dos valores considerados, e um elemento terminal Z representando a stack vazia. Ou seja,

$$C(a_1) \frown C(a_2) \frown C(a_3) \frown ... \frown C(a_n) \frown Z$$

Vejamos, por fim, e como exemplo, o efeito da realização de um $\overline{pop}$ na stack representada por $C(a) \frown C(b) \frown Z$. Temos, pois,

$$
\begin{aligned}
C(a) \frown C(b) \frown Z &\xrightarrow{\overline{pop}\langle a\rangle} (o(x).C(x) + \overline{e}.Z) \frown C(b) \frown Z \\
&\xrightarrow{\tau} C(b) \frown (o(x).C(x) + \overline{e}.Z) \frown Z \\
&\xrightarrow{\tau} C(b) \frown Z \frown Z \\
&= C(b) \frown Z
\end{aligned}
$$

Como exercício, será conveniente mostrar que esta implementação é observacionalmente congruente com a especificação $St(s)$, para toda a sequência s (possivelmente necessitará de recorrer a um argumento indutivo sobre o comprimento de s).

4 Sistemas de Estrutura "Indutiva"

Vamos agora abordar um tipo de implementações que apresentam uma estrutura *fixa* mas de tamanho *arbitrário*. Tais estruturas são classificados como *indutivas* porque um sistema de tamanho $n+1$ pode ser definido à custa de um sistema similar de tamanho n.

Considere-se um processo que recebe numa porta in uma sequência de n números e fornece, através de uma porta $\overline{out}$, essa mesma sequência ordenada por ordem decrescente terminada por um 0.

$$
\begin{aligned}
Ord_n &\triangleq in(x_1).in(x_2).....in(x_n).P(\{x_1, x_2, ..., x_n\}) \\
P(S) &\triangleq \overline{out}\langle max(S)\rangle.P(S - \{max(S)\}) \\
P(\emptyset) &\triangleq \overline{out}\langle 0\rangle.Ord_n
\end{aligned}
$$

Vamos implementar um ordenador de n números por agregação de n células C capazes de armazenarem dois números e os comparar. As componentes são genéricas, no sentido em que a sua dinâmica não depende do conhecimento do tamanho do ordenador de que fazem parte. O esquema seguido é semelhante ao dos exemplos discutidos anteriormente. Assim, cada C dispõe de duas portas auxiliares u e $\overline{d}$ que se ligam, respectivamente, às portas in e $\overline{out}$ da célula adjacente. A estrutura dessa ligação é, de novo, captada por um operador $\frown$ cuja definição detalhada é deixada ao leitor. Como nos casos do contador e da stack, existe um processo tampão Z que fecha a cadeia e que, consequentemente, não dispõe das portas auxiliares.

A principal diferença entre este e os exemplos anteriores reside no facto do tamanho da cadeia de células C ser, agora, previamente fixado pelo implementador. Note-se porém que, dado um ordenador de n números

$$
Ord^n \triangleq C \frown C \frown C \frown ... \frown B
$$

um ordenador para $n+1$ números pode ser obtido simplesmente por agregação de uma nova componente C, *i.e.*,,

$$
Ord^{n+1} \triangleq C \frown C^n
$$

As componentes base C e Z definem-se por

$$
\begin{aligned}
C &\triangleq in(x).D(x) \\
D(x) &\triangleq \overline{d}(x).C + u(y).\overline{out}\langle max(\{x,y\})\rangle.E(min(\{x,y\})) \\
E(x) &\triangleq \mathsf{if}\ x = 0\ \mathsf{then}\ \overline{out}\langle 0\rangle.C\ \mathsf{else}\ D(x) \\
&\text{e} \\
Z &\triangleq \overline{out}\langle 0\rangle.Z
\end{aligned}
$$

O argumento de correcção — *i.e.*,, a prova de que $Ord_n = Ord^n$, para todo o $n \geq 0$ — é similar ao dos casos anteriores.

Referências

[Fen96] C. Fencott. *Formal Methods for Concurrency*. International Thomson Computer Press, 1996.

[Mil89] R. Milner. *Communication and Concurrency*. Series in Computer Science. Prentice-Hall International, 1989.

[Mil99] R. Milner. *Communicating and Mobile Processes: the π-Calculus*. Cambridge University Press, 1999.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (parts i and ii). *Information and Computation*, (100):1–77, 1992.

[Pet62] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Technische Hochschule Darmstadt, 1962.

A Exercícios

Exercício 1

Reporte-se à especificação distribuída de uma *stack* discutida nas aulas. Mostre que definição proposta é observacionalmente congruente com a especificação St_s, para toda a sequência s.

Exercício 2

Reporte-se ao exemplo de um sistema de estrutura 'indutiva' discutido nas aulas. Verifique detalhadamente o correspondente argumento de correcção mostrando que, para todo o $n \geq 0$, se tem $Ord_n = Ord^n$.

Exercício 3

Um *broadcaster* de grau n, para $n > 2$, é um processo designado por B_n, com uma porta a e n portas etiquetadas por $\overline{b_1}$ a $\overline{b_n}$. O seu comportamento consiste em aguardar a ocorrência de a e activar seguidamente, e por uma ordem arbitrária, as portas $\overline{b_i}$. Após isso não realiza mais acções.

1. Defina este processo sem recorrer ao operador $+$.
2. Defina um combinador $\frown$ em termos dos operadores estáticos tal que, para todo o $n > 2$, $B_{n-1} \frown B_2 = B_n$.
3. Prove essa igualdade para $n = 3$.
4. Esboce uma prova da mesma igualdade para um n qualquer.

Exercício 4

Considere o processo $B \triangleq b.c.B$ a partir do qual se definiu

$$S \triangleq [P \frown B \frown B]$$

onde

$$[P \frown Q \frown R] \overset{\text{abv}}{=} \mathsf{new}\,\{b_1, b_2\}\ (P \mid \{f_1\}\,Q \mid \{f_2\}\,R)$$

e cada f_i renomeia b para b_i e c para c_i, para $i \in \{1, 2\}$. Suponha que P é um processo ainda não especificado mas tal que $\mathsf{fn}(P) = \{a, \overline{b_1}, \overline{b_2}\}$.

1. Esboce o diagrama de sincronização de S.
2. Suponha que o comportamento desejado para S é dado pela seguinte equação:

   $$S \approx a.(c_1.c_2.\,\mathbf{0} + c_2.c_1.\mathbf{0})$$

   Mostre que, fazendo $P \triangleq a.\overline{b_1}.\overline{b_2}.\mathbf{0}$, S verifica esta equação.
3. Existem outras possibilidades para a definição de P tais que a equação acima ainda é verificada. Indique duas dessas possibilidades que não sejam observacionalmente equivalentes nem entre si nem a $a.\overline{b_1}.\overline{b_2}.\mathbf{0}$.

Exercício 5

Considere a seguinte especificação de um controlador que visa assegurar a execução repetida de uma determinada tarefa por cada um de n processos P_1, P_2, ..., P_n. Cada processo P_i inicia a tarefa através de um pedido reconhecido pelo controlador como a acção rq_i e sinaliza o seu fim através da acção reconhecida como tr_i. O controlador garante que as acções rq_i devem ocorrer ciclicamente, começando em rq_1, e que, para cada i, as acções rq_i e tr_i ocorrem de forma alternada, começando com rq_i.

A especificação abaixo é feita em termos de uma família de processos $\mathsf{Ct}_{i,X}$ parameterizados por um número i e um conjunto X. Intuitivamente, o parâmetro i indica ser a vez do processo P_i iniciar a sua execução, enquanto o parâmetro X indica o conjunto de processos que estão correntemente em execução.

$$\mathsf{Control} \;=\; \mathsf{Ct}_{1,\emptyset}$$

$$\begin{aligned}\mathsf{Ct}_{i,X} \;=\; & \mathsf{if}\;\; i \in X \\ & \quad \mathsf{then}\;\; \sum_{k\in X} tr_k \cdot \mathsf{Ct}_{i,X-\{k\}} \\ & \quad \mathsf{else}\;\; \sum_{k\in X} tr_k \cdot \mathsf{Ct}_{i,X-\{k\}} + rq_i \cdot \mathsf{Ct}_{i+1,X\cup\{i\}}\end{aligned}$$

Suponha, agora, que se pretende realizar o controlador através de uma cadeia ligada de n células idênticas A_i definidas por

$$\mathsf{A}_i \;=\; rq_i \cdot x_i \cdot (tr_i \cdot \overline{x}_{i-1} \cdot \mathsf{A}_i \;+\; \overline{x}_{i-1} \cdot tr_i \cdot \mathsf{A}_i)$$

Cada célula tem pois duas portas (rq_i e tr_i) para controlar o processo P_i, assim como outras duas portas para se ligar em anel fechado. O controlador é então dado por

$$\mathsf{Control}' \;=\; \mathsf{new}\,\{x_i \mid 1 \le i \le n\}\; (\mathsf{A}_1 \mid \overline{x}_1 \cdot \mathsf{A}_2 \mid \overline{x}_2 \cdot \mathsf{A}_3 \mid \ldots \mid \overline{x}_{n-1} \cdot \mathsf{A}_n)$$

1. Esboce o diagrama de sincronização de Control'
2. Calcule a sua espécie sintática
3. Desenhe os grafos de transição de Control e Control' para $n = 2$.
4. Para $n = 3$ mostre que $\mathsf{Control} \approx \mathsf{Control}'$
5. Indique como generalizaria a prova anterior para n arbitrário.
6. Suponha que alguém propôs uma versão simplificada das células A_i:

   $$\mathsf{A}_i \;=\; rq_i \cdot x_i \cdot tr_i \cdot \overline{x}_{i-1} \cdot \mathsf{A}_i$$

   Averigue se, usando, na especificação de Control′ esta definição das células, é ou não ainda possível estabelecer a equivalência $\mathsf{Control}' \approx \mathsf{Control}$.

Exercício 6

Um *duplicador* é um processo P que aceita um valor na porta e e o transmite, por ordem arbitrária, nas portas $\overline{a}_1$ e $\overline{a}_2$, de acordo com a definição seguinte:

$$P \;\triangleq\; e(x).(\overline{a}_1 x.\mathbf{0} \mid \overline{a}_2 x.\mathbf{0})$$

1. Suponha que alguém apresentou a seguinte especificação de um duplicador cíclico, *i.e.*,, que retorna ao estado inicial após ter completado um ciclo de operação (ou seja após ter recebido e transmitido x nas duas portas de saída):

   $$P \;\triangleq\; e(x).(\overline{a}_1 x.P \mid \overline{a}_2 x.P)$$

   Explique porque razão esta sugestão está errada e forneça a definição correcta.
2. Considere os seguintes processos

   $$\begin{aligned}P_1 &\triangleq \mathsf{new}\,\{m\}\;(\{t_1/a_1, m/a_2\}P \mid \{m/e, t_2/a_1, t_3/a_2\}P)\\ P_2 &\triangleq \mathsf{new}\,\{m\}\;(\{m/a_1, t_2/a_2\}P \mid \{m/e, t_1/a_1, t_3/a_2\}P)\end{aligned}$$

   Esboce os diagramas de sincronização de P_1 e P_2.
3. Mostre que se na construção de P_1 e P_2 forem usados apenas duplicadores não cíclicos, se tem $P_1 \approx P_2$.
4. Será que o mesmo acontece caso se utilizem apenas duplicadores cíclicos? Porquê?
5. Explique de que modo poderia usar o CWB para responder às 4 alíneas anteriores.

Exercício 7

Considere o problema de modelar na linguagem de processos que estudou, o dispositivo usado em sorteios do tipo *totoloto*. Suponha dado um conjunto de três 'bolas', $\{b_1, b_2, b_3\}$. O dispositivo começa por selecionar aleatoriamente uma delas, processo que se repete sucessivamente. Recorrendo à acção τ para representar a actividade interna do dispositivo, uma possível especificação do comportamento será a seguinte:

$$D \triangleq \tau.b_1.D + \tau.b_2.D + \tau.b_3.D$$

1. Mostre que a selecção da 'bola' b_2 é possível através da identificação e verificação da transição correspondente no processo D.
2. Considere, agora, uma possível implementação deste dispositivo com base numa célula C definida por:

$$C \triangleq start.Move$$
$$Move \triangleq \tau.Eject + \overline{go}.C$$
$$Eject \triangleq b.Move$$

   onde $start$ modela a acção de activação externa da célula, $\overline{go}$ a activação da célula vizinha à sua direita e b a selecção da bola b. A implementação do dispositivo de sorteio vem, assim,

$$E_1 \triangleq \{x_1/start, b_1/b, x_2/go\}\ Eject$$
$$C_2 \triangleq \{x_2/start, b_2/b, x_3/go\}\ C$$
$$C_3 \triangleq \{x_3/start, b_3/b, x_1/go\}\ C$$

$$S \triangleq \mathsf{new}\ \{x_1, x_2, x_3\}\ (E_1 \mid C_2 \mid C_3)$$

   (a) Esboce o grafo de transições do processo S

   (b) Como se modificaria o grafo de transições de S se a definição de $Eject$ fosse substituida por:

$$Eject \triangleq b.\overline{go}.C$$

   (c) Prove ou refute a equivalência $D \approx S$.

3. Note que os processos D e S apresentam um comportamento perpétuo. Uma maneira de limitar esse comportamento consiste em encarar o próprio 'saco de bolas' como um processo $Saco$, com espécie $\{\overline{b}_1, \overline{b}_2, \overline{b}_3\}$ em que cada acção $\overline{b}_i$ modelasse a retirada da bola correspondente.

   (a) Defina este processo $Saco$ de modo a garantir que as bolas são retiradas aleatoriamente e que cada acção $\overline{b}_i$ é executada apenas uma vez.

   (b) Mostre que o processo $DS = \mathsf{new}\ \{b_1, b_2, b_3\}\ (D \mid Saco)$ atinge estados a partir dos quais nenhuma transição pode ocorrer.

   (c) O processo DS não exibe, porém, qualquer comportamento visível. Explique porquê e introduza as modificações necessárias para que os eventos b_1, b_2 e b_3 continuem a ser observáveis externamente.