CCS 1: Modelação de Processos em CCS

Luís Soares Barbosa

Sumário

Este texto inicia-se com a introdução da noção de processo definindo-a, informalmente, como o padrão de evolução das capacidades de interacção de um sistema de transição. Introduz-se, de seguida, a sintaxe e a semântica operacional de uma linguagem para descrição de processos proposta por Robin Milner há cerca de 30 anos atrás e popularizada sob a designação CCS*. A linguagem inclui, entre outros, operadores de composição paralela e escolha não determinística entre diferentes possibilidades de evolução.*

O uso desta notação é ilustrado através da discussão detalhada de vários exemplos de modelação.

1 Sistemas de Transição e Processos

Aquilo que é externamente observável num sistema de transição é o padrão de evolução das suas capacidades de interacção, que designamos por *comportamento*. Dizemos, por exemplo, que o sistema *interage através do nome* (acção, porta, canal) a e, a partir desse ponto, comporta-se desta ou daquela forma. Do mesmo modo, recorde-se, dizíamos que um autómato pode, a partir de um dado estado, reconhecer o símbolo a e prosseguir para outro estado.

Podemos, assim, abstrair toda uma família de sistemas num particular sistema de transição cujos estados correspondem a comportamentos e as transições a alterações de comportamento disparadas pela ocorrência de acções[1]. A cada um desses estados chamaremos um *processo*.

Processo $\Leftrightarrow$ Padrão de evolução das capacidades de interacção de um sistema

Processos são, pois, construidos através da composição de possibilidades ou capacidades de interacção. I.e., de eventos que ocorrem autonomamente, de forma atómica (*i.e.*, indivisível no tempo), ao nível de abstracção considerado, e que são susceptíveis de serem observados (por outras componentes do sistema ou pelo seu ambiente). Em álgebra de processos (*cf.*, [Hoa85, Hen88, Mil89]) a designação *acção* é usada, para sublinhar o caracter atómico e não interpretado do conceito. Trata-se de uma possibilidade de comunicação, de um canal, de uma porta, mais genericamente ainda, de uma *ligação*, ou *adjacência*, entre entidades capazes de interagir através dela. Assim,

Acção $\Leftrightarrow$ Possibilidade de interacção

[1] Em termos coalgébricos, recordando a Lição anterior, tal sistema corresponde à coalgebra *final* para o functor indicado [Rut00].

A disciplina que governa a interação entre processos é modelada formalmente por uma álgebra sobre o conjunto das acções. Álgebra significa aqui, como sempre, uma coisa muito simples: uma operação que especifica qual o resultado da ocorrência conjunta de duas acções.

Apesar de existirem, na prática, diversos regimes de comunicação, consideraremos neste curso uma noção de interacção muito simples, a partir da qual, de resto, outras podem ser modeladas. Trata-se do *handshake* binário, *i.e.*, a transferência instantânea de informação entre apenas dois processos. Dizemos que dois processos *sincronizam* (ou *interagem*) quando realizam simultaneamente um par de acções *complementares*. Acções complementares representam-se por símbolos idênticos a menos de uma barra horizontal escrita por cima, por exemplo a e $\overline{a}$. O resultado de qualquer sincronização é uma acção não observável que, na notação que iremos adoptar, se representa pela letra τ. Formalmente,

Definição 1 *Seja L um conjunto de nomes, ou etiquetas. A partir dele definimos o conjunto Act das acções como*

$$Act \quad ::= \quad a \mid \overline{a} \mid \tau$$

para todo o $a \in L$.

Começamos, assim, a falar de notações. Em verdade, se formalmente, um processo é a denotação matemática de um comportamento, é também verdade que necessitamos de uma *linguagem* em que os possamos representar. Por um ligeiro abuso de linguagem, usaremos também a palavra *processo* para nos referirmos a um termo numa linguagem que iremos designar por $\mathbb{P}$, introduzida por R. Milner há cerca de 30 anos [Mil89]. Repare-se que a situação é similar à associação de uma função a um termo em notação λ. Antes, porém , de introduzirmos a linguagem de modelação a adoptar, comecemos por a ilustrar considerando alguns exemplos.

2 Primeiros Exemplos

Buffers Simples

Buffer de uma posição: $A(in, out) \triangleq in.\overline{out}.\mathbf{0}$

... permanente: $B(in, out) \triangleq in.\overline{out}.B$

... com duas saídas: $C(in, o_1, o_2) \triangleq in.(\overline{o_1}.C + \overline{o_2}.C)$

... não determinístico: $D(in, o_1, o_2) \triangleq in.\overline{o_1}.D + in.\overline{o_2}.D$

Repare-se como na declaração de um processo se enumeram as suas portas de comunicação (ou acções), sem contudo indicar a respectiva polaridade (o que é feito apenas no corpo da definição através da distinção entre a e $\overline{a}$). É, contudo, usual, para simplificar a escrita, omitir a enumeração das portas escrevendo, por exemplo, A em vez de $A(in, out)$.

Parametrização

Consideremos

$$B(in, out) \triangleq in(x).\overline{out}\langle x\rangle.B$$

Podemos encarar $in(x)$ como uma interacção em que um valor é recebido na porta in e associado à variável x. Similarmente, $\overline{out}\langle x\rangle$ representa uma interacção em que o valor associado a x é comunicado (para o exterior) através da porta $\overline{out}$.

Tal como acima introduzimos parametrização nas acções, é, por vezes, útil fazê-lo na própria definição de um processo. Seja, por exemplo, a seguinte especificação de um *buffer* de n-posições (em que n é um número natural arbitrário mas fixo):

$$\begin{aligned} Bn_{\epsilon} &\triangleq in(x).Bn_x \\ Bn_{s:u} &\triangleq \overline{out}\langle u\rangle.Bn_s && \text{se } \mathsf{len}(s) = n-1 \\ Bn_{s:u} &\triangleq in(x).Bn_{x:s:u} + \overline{out}\langle u\rangle.Bn_s && \text{se } \mathsf{len}(s) < n-1 \end{aligned}$$

NOTA: Neste curso encontraremos vários exemplos de processos que manipulam dados de diversos tipos (naturais, conjuntos, sequências, etc.). Estes domínios de dados, assim como as operações sobre eles (*e.g.*, adição, concatenação, intersecção, etc.) *não* são específicos da linguagem de processos que estamos a introduzir. Vamos assumir que o seu significado é conhecido do leitor e remeter a sua especificação para outras disciplinas do curso onde se estuda a definição matemática de tipos de dados. A ideia fundamental na linguagem e no cálculo de processos que aqui nos ocupa é de que, mesmo sem nos preocuparmos com estruturas de dados elaboradas, é possível raciocionar sobre o *comportamento* dos processos e inferir propriedades importantes sobre ele (por exemplo, ausência de *deadlock*).

Para se perceber o exemplo anterior, convencionamos designar por ϵ a sequência vazia, representar pelo mesmo símbolo o elemento x e a sequência unitária por ele formada, utilizar os símbolos : e len para representar, respectivamente, a concatenação de duas sequências não vazia e o comprimento de uma sequência.

A parametrização oferece um modo mais intuitivo de especificar alguns processos, mas, como veremos adiante, *não* aumenta o poder expressivo da linguagem. Por agora, basta sublinhar que podemos pensar por exemplo na definição $A_x \triangleq \overline{a}(x).A'$, onde x percorre os naturais, como uma *abreviatura* do seguinte conjunto (infinito) de definições:

$$\begin{aligned} A_0 &\triangleq \overline{a_0}.A' \\ A_1 &\triangleq \overline{a_1}.A' \\ A_2 &\triangleq \overline{a_2}.A' \\ A_3 &\triangleq \overline{a_3}.A' \\ &\ldots\ldots \end{aligned}$$

No exemplo Bn a especificação apresentada traduz-se numa definição para cada possível parâmetro (*i.e.*, para cada sequência de naturais de 0 a n). Note-se, ainda, que as condições laterais (*e.g.*, $\mathsf{len}(s) < n-1$) não fazem formalmente parte da definição. Apenas estabelecem quais os valores que podemos tomar de modo a que a definição em causa possa ser aplicada. No caso de Bn a expressão

$$Bn_{s:u} \triangleq in(x).Bn_{x:s:u} + \overline{out}\langle u\rangle.Bn_s \qquad \text{se } \mathsf{len}(s) < n-1$$

significa que dispomos de uma definição para cada caso em que o parâmetro é uma sequência não vazia de comprimento inferior a n.

Composição Paralela

Um meio de, dado um *buffer* de 1 posição, obter um *buffer* de, por exemplo, 2 posições, consiste em colocar dois processos B em paralelo, *i.e.*, activados concorrentemente, e adequadamente ligados de forma a assegurar que os valores presentes à entrada do primeiro fluam para o segundo

e, deste, para o exterior. Os exemplos seguintes motivam a introdução de dois operadores fundamentais na linguagem: *composição paralela* e *restrição*.

Buffer com 2 posições: vamos colocar dois buffers em paralelo ligando a porta $\overline{out}$ do primeiro à porta in do segundo. Definimos primeiro cópias de B, renomeando as portas de forma adequada. O seu objectivo é assegurar que a acção $\overline{out}$ do primeiro *buffer* se torne complementar da acção in do segundo, estabelecendo uma possibilidade de comunicação. As duas componentes são, então, compostas concorrentemente e a interacção, através da porta m é tornada interna, por restricção. Esta internalização significa que as interacções m e $\overline{m}$ deixam de estar disponíveis para o exterior. Assim,

$$S \triangleq \mathsf{new}\,\{m\}\ (B\langle in, m\rangle \mid B\langle m, out\rangle)$$

Note-se que $B\langle in, m\rangle$ corresponde, pelo mecanismo usual de substituição de variáveis num termo, a $\{m/out\}\, B\langle in, out\rangle$.

Buffer com n posições: elaborando sobre o esquema anterior chegamos a

$$Bn' \triangleq \mathsf{new}\,\{m_i \mid i < n\}\ (B\langle in, m_1\rangle \mid B\langle m_1, m_2\rangle \mid \cdots \mid B\langle m_{n-1}, out\rangle)$$

Exclusão Mútua Seja $\{P_i \mid i \in I\}$ um conjunto de processos cada um com uma região crítica representada pela acção c_i e cuja activação em simultâneo se pretende evitar. O problema é resolvido por recurso a um semáforo (modelado pelo processo Sem) ao qual é solicitado acesso sempre que um processo decide entrar na sua região crítica. Assim,

$$\begin{aligned} Sem &\triangleq get.put.Sem \\ P_i &\triangleq \overline{get}.c_i.\overline{put}.P_i \\ S &\triangleq \mathsf{new}\,\{get, put\}\ (Sem \mid (|_{i\in I}\ P_i)) \end{aligned}$$

3 Uma Linguagem de Processos

Após termos encontrado algumas especificações de processos muito simples, três questões estão, por certo, na mente do leitor:

- Qual é a sintaxe exacta da linguagem usada?
- Qual é o *significado* preciso destas especificações?
- Porque razão algumas das nossas construções favoritas (*e.g.*, atribuições, ciclos, ...) não fazem parte da linguagem?

Na tentativa de responder a estas questões, vamos começar por detalhar o significado intuitivo associado aos construtores de processos que encontramos. Apresentaremos, de seguida, a sintaxe da linguagem base e abordaremos a sua semântica na secção seguinte.

NOTA: Relativamente à terceira questão diremos que a razão principal para certas omissões é a procura de simplicidade. De facto, a teoria e o cálculo associado à linguagem de processos exige-nos um compromisso entre *poder expressivo* e *tratabilidade* (formal). Uma linguagem excessivamente pobre (por exemplo, sem um operador de composição) facilitaria muito o desenvolvimento da teoria (*e.g.*, originando provas muito mais concisas) mas tornaria o cálculo razoavelmente inútil. Por outro lado, uma linguagem muito diversificada em construções sintáticas complicaria desnecessariamente a teoria. Isto explica a razão pela qual muitas construções interessantes, mas que podem ser definidas à custa dos operadores mais primitivos (por exemplo, atribuições) são omitidas. Quando delas necessitarmos podemos usa-las como *macros*, sem ter necessidade de modificar as definições básicas.

Se este sentido de economia dita a concisão da linguagem, não explica porque foram estas e não outras as escolhas feitas. Aqui, e de uma forma geral, prevaleceu a orientação seguinte: o cálculo apenas quer tratar um aspecto dos processos, a saber, os padrões das suas evoluções por realização de acções. Deliberadamente se exclui, portanto, informação sobre, por exemplo, relações de causalidade entre as acções ou a sua duração. À luz deste critério percebe-se, por exemplo, a razão pela qual a noção de temporizador, básica nos sistemas de tempo real, não é aqui considerada. Existem, no entanto, diversas extensões ao cálculo que acomodam considerações sobre tempo real [Sch90, BB91, Fen96]. Para o leitor interessado, [Mil86] contém uma exposição brilhante sobre os princípios adoptados na concepção de CCS e possíveis alternativas.

Operadores Estáticos

São operadores que persistem (como conectivos principais) ao longo da vida do processo. Servem para definir uma estrutura base de conexão que não é alterada pela ocorrência de acções.

Paralelo: $P \mid Q$, coloca os processos P e Q em paralelo, interagindo através das portas complementares. Por abreviatura, podemos usar uma forma iterada deste operador para um conjunto de indexação finito, por exemplo em $|_{i\in\{1,2,3,4\}}\ P_i$, que abrevia $P_1 \mid P_2 \mid P_3 \mid P_4$.

Restricção: new $K\ P$, impossibilita a interacção com P através do conjunto K de nomes. Estes são ditos *locais*, ou internos, a P.

Assume-se, ainda, a usual operação de *substituição* sintática sobre o conjunto L dos nomes

$$\{\sigma\}\, P$$

a que se recorre para renomear as portas de P de acordo com a substituição σ. Por exemplo,

$$\{c/b\}\, a.b.\mathbf{0}$$

Alternativamente, pode ser introduzido no cálculo um operador de renomeação explícito (*e.g.*, $P[\sigma]$). Este procedimento é adoptado em [Mil89] e, regra geral, nas ferramentas de análise de processos (por exemplo, no `cwb-nc`).

Operadores Dinâmicos

Tratam-se operadores que definem comportamentos sequenciais não deterministas.

Inação: $\mathbf{0}$, representa o processo inactivo (inerte), incapaz de qualquer tipo de interação.

Prefixação: $a.P$, realiza a e comporta-se, de seguida, como P. Note-se que o primeiro argumento é um elemento do conjunto Act (e não um processo), pelo que a prefixação não é um conectivo de composição sequencial de processos.

Escolha: $P{+}Q$, comporta-se como P ou como Q. Assumindo que o par $\langle \mathbf{0}, + \rangle$ forma um monoide em $\mathbb{P}$, a escolha admite uma forma iterada, obtida por redução monoidal: $\sum_{i\in I} P_i$, onde I é um conjunto de indexação, ou abreviadamente, $\sum \tilde{P}$, sendo $\tilde{P} \stackrel{\text{abv}}{=} \{P_i \mid i \in I\}$ uma família indexada de processos. Claramente, $\sum_{i\in\emptyset} P_i = \mathbf{0}$.

Alternativamente (por exemplo em [Mil99]) processos sequenciais são definidos por termos da forma $\sum_{i\in I} a_i.E_i$, com $a_i \in Act$. Nesse caso tanto $\mathbf{0}$ como a escolha binária são definidos por abreviatura.

Sintaxe

Seja A um conjunto de *nomes*, ou *etiquetas* (de acções) e $\overline{A}$ um conjunto de *co-nomes* (i.é, de acções complementares). Designamos por L o conjunto $A \cup \overline{A}$ e por $Act = L \cup \{\tau\}$. Designamos por $\mathbb{P}$ o conjunto dos termos gerados pelo seguinte BNF:

$$E \ ::= \ A(x_1, ..., x_n) \ \mid \ a.E \ \mid \ \sum_{i \in I} E_i \ \mid \ E_0 \mid E_1 \ \mid \ \mathsf{new}\, K \ E \tag{1}$$

(onde $a \in Act$ e $K \subseteq L$) a que nos referiremos por *processos*. A forma $A(x_1, ..., x_n)$, onde A varia sobre um conjunto de identificadores de processos e x_i sobre um conjunto de acções em L, introduz a declaração de um processo A com potencialidade de interacção nas portas x_1 a x_n. A todo o identificador de processo corresponde uma equação de definição da forma

$$A(\tilde{x}) \ \triangleq \ E_A \tag{2}$$

com $\mathsf{fn}(E_A) \subseteq \tilde{x}$, onde $\mathsf{fn}(P)$ designa o conjunto das variáveis livres de P. Atente-se no modo como os nomes que parametrizam um processo são instanciados na sua invocação. Por exemplo,

$$A(a, b, c) \ \triangleq \ a.b.\mathbf{0} + c.A\langle d, e, f\rangle$$

Por convenção, a invocação de A sem explicitar o tuplo de nomes fornecidos como parâmetros actuais, corresponde a uma invocação em que esses nomes são idênticos aos que constam na declaração de A como parâmetros formais: neste caso a $A\langle a, b, c\rangle$. Um modo alternativo de introduzir definições recursivas é através da construção $\underline{fix}\ (X = E_X)$, que evita a multiplicação de identificadores tornando o cálculo mais expedito.

Por convenção daremos maior precedência à prefixação, seguida da restricção, relativamente aos restantes operadores. Note-se que a escolha binária e a inacção são definidas por abreviatura, respectivamente,

$$E_0 + E_1 \stackrel{\text{abv}}{=} \sum_{i \in \{0,1\}} E_i \tag{3}$$

$$\mathbf{0} \stackrel{\text{abv}}{=} \sum_{i \in \emptyset} E_i \tag{4}$$

Sobre os termos desta linguagem aplicam-se da forma usual *substituições* sintáticas — por exemplo em $\{c/b\}\ a.b.\mathbf{0}$. Da mesma forma se recorre, sempre que necessário, à renomeação de variáveis mudas (*cf.*, a conversão-α estudada no λ-calculus). Por exemplo, $\{x/y\}\ \mathsf{new}\ \{x\}\ y.x.\mathbf{0} \ = \ \mathsf{new}\ \{x'\}\ x.x'.\mathbf{0}$

NOTA: A introdução de τ para representar uma acção *interna* (*i.e.*, não observável) é fonte de algumas confusões no leitor menos avisado. O cerne da questão reside no facto de, apesar de não observável, esta acção não poder ser ignorada. Considere-se, por exemplo, o processo

$$P \triangleq a.P + \tau.b.P$$

Se fosse possível ignorar τ, este processo exibiria um comportamento similar a

$$Q \triangleq a.Q + b.Q$$

No entanto, enquanto em Q a escolha entre a realização de a ou b é totalmente controlável pelo exterior, e, portanto, determinista, em P o processo pode evoluir silenciosamente para um estado em que apenas b é oferecida. A ocorrência de τ é autónoma, no sentido em que não requer nem necessita de qualquer participação externa.

Em geral, como veremos mais tarde, não podemos afirmar a equivalência entre um processo T e $\tau.T$, captando assim um facto (não raro indesejável) que muitas vezes está presente nos sistemas concorrentes reais. O facto de não ser possível ignorar as acções internas deve-nos fazer desde já suspeitar que a relação de equivalência entre processos deve ser mais complexa do que a clássica equivalência entre autómatos (porquê?).

Espécie

A *espécie* de um processo P representa as suas capacidades de interacção, *i.e.*, a sua interface com o ambiente. Em rigor, diz-se que $K \subseteq L$ constitui uma espécie de P — e escreve-se $P : K$ — sse todas as acções que P pode realizar ao longo da sua "vida" (*i.e.*, todas as acções de P e de todas as suas derivações) estiverem contidas em K. Claramente, se $P : K$ e $K \subseteq K'$, então $P : K'$. Uma espécie é dita *mínima* se contiver essas e apenas essas acções.

A importância desta noção reside no facto de algumas leis do cálculo dependerem de condições sobre as espécies dos processos envolvidos. Por exemplo, a cláusula seguinte, usada na definição da congruência estrutural,

$$\mathsf{new}\ \{a\}\ (P \mid Q) \ \Leftrightarrow\ \mathsf{new}\ \{a\}\ P \mid Q$$

só é válida se a e $\overline{a}$ não pertencerem à espécie do processo Q, caso em que a restrição pode ser 'passada para dentro'.

A espécie mínima de um processo P — *i.e.*, $\bigcap\{K \subseteq L | P : K\}$ — nem sempre é fácil de determinar. De facto, mesmo que um dado $a \in L$ ocorra sintaticamente na expressão de P, pode não ser trivial verificar se P vai ou não realiza-la. O problema da determinação da espécie mínima envolve o cálculo da árvore de derivação do processo e é, de resto, e no caso geral, indecidível.

Uma boa aproximação à espécie mínima de um processo P é dada pelo respectivo conjunto de variáveis livres $\mathsf{fn}(P)$, por vezes designado por *espécie sintática* de P. Este é simples de calcular:

$$\mathsf{fn}(a.P) = \{a\} \cup \mathsf{fn}(P) \tag{5}$$
$$\mathsf{fn}(\tau.P) = \mathsf{fn}(P) \tag{6}$$
$$\mathsf{fn}(\sum_{i\in I} P_i) = \bigcup_{i\in I} \mathsf{fn}(P_i) \tag{7}$$
$$\mathsf{fn}(P \mid Q) = \mathsf{fn}(P) \cup \mathsf{fn}(Q) \tag{8}$$
$$\mathsf{fn}(\mathsf{new}\ K\ P) = \mathsf{fn}(P) - (K \cup \overline{K}) \tag{9}$$

e, para cada $P(\tilde{x}) \triangleq E$, $\mathsf{fn}(E) \subseteq \mathsf{fn}(P(\tilde{x})) = \tilde{x}$. Claramente, para uma substituição σ, tem-se

$$\mathsf{fn}(\{\sigma\}\ P) = \{\sigma\, a \mid a \in \mathsf{fn}(P)\} \tag{10}$$

Note-se, no entanto, que nem sempre a espécie sintática é mínima. Por exemplo, o processo $\mathsf{new}\ \{a\}\ (a.b.c.0)$ não tem transições, logo a sua espécie mínima é $\emptyset$, embora $\mathsf{fn}((\mathsf{new}\ \{a\}\ a.b.c.0)) = \{b, c\}$.

4 Diagramas

Da expressão que define um processo é possível inferir dois diagramas que se revelam muito úteis na sua análise. O primeiro representa as capacidades de interacção do processo e a estrutura das suas conexões internas, fornecendo uma visão *espacial* do processo. É, por isso, designado por *diagrama de sincronização*. O segundo representa as possíveis evoluções, passo a passo, do processo através da realização das interacções disponíveis. Esta visão *temporal* do processo é sugestivamente designada por *grafo de transições*. Assim,

Diagrama de Sincronização

Neste diagrama os nodos representam (sub-)processos e consistem em pequenas circunferências sobre as quais se marcam as portas de interacção disponíveis. Os nomes internos dessas portas anotam-se na região interior da circunferência; nomes externos (resultantes de substituições)

anotam-se paralelamente aos anteriores mas no exterior da circunferência. Portas complementares são ligadas por arcos. Note-se que os construtores *estáticos* de processos podem ser interpretados como operadores de uma álgebra de diagramas de sincronização. Assim, a composição paralela traça um arco entre portas externas complementares, enquanto a restricção a K remove os nomes externos a e $\overline{a}$ para cada $a \in K$. Por seu lado, as substituições atribuiem novas etiquetas externas às portas de interacção.

Grafo de Transições

Como vimos, um processo evolui transitando de uma configuração (ou *estado*) para outra sempre que realiza uma acção. Como as várias configurações, ou estados, resultantes dessa evolução são representadas por expressões em $\mathbb{P}$, podemos escrever

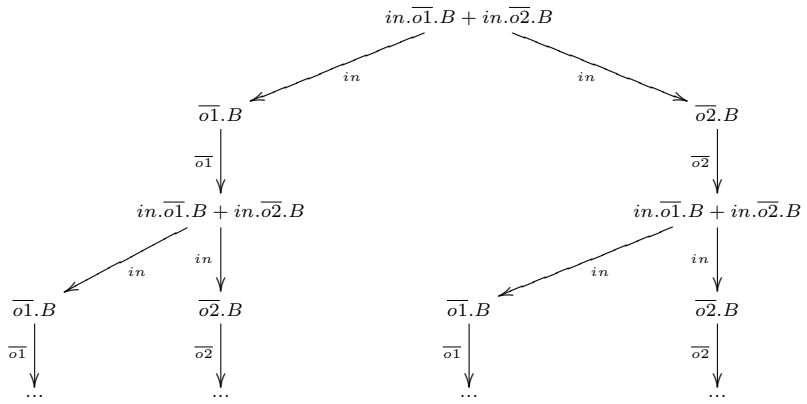
$$P \xrightarrow{a} P' \tag{11}$$

para nos referirmos a uma transição (elementar) de P para P' através da realização de a. Neste caso chamamos ao par (a, P') uma *derivação imediata* de P e a P' uma *a-derivação* de P. Da mesma forma, numa sequência de transições

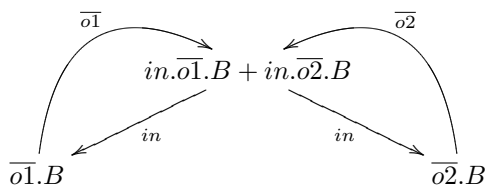
$$P \xrightarrow{a_1} P_1 \xrightarrow{a_2} P_2 \xrightarrow{a_3} ... \xrightarrow{a_n} P' \tag{12}$$

designamos o par $(a_1a_2a_3...a_n, P')$ por *derivação* de P e referimo-nos a P' como uma $a_1a_2a_3...a_n$-*derivação* de P.

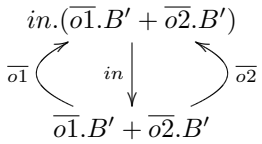
Uma maneira de esboçar o comportamento de um processo P consiste em agrupar numa árvore todas as suas derivações, as derivações das suas derivações e assim sucessivamente. A estrutura resultante designa-se por *árvore de derivação*. Por exemplo, para $B \triangleq in.\overline{o1}.B + in.\overline{o2}.B$ temos,



A partir da *árvore de derivação* podemos construir o *Grafo de Transições* colapsando todos os nodos etiquetados pela mesma expressão num único nodo. Desta forma o comportamento do processo P é representado por um grafo cujos nodos são expressões derivadas de P e os arcos, etiquetados por acções, representam a possibilidade de o processo representado no nodo origem evoluir para a configuração no nodo destino através da realização da acção que etiqueta o arco. Para o mesmo exemplo, obtemos,



Se compararmos o grafo anterior com o que se obtém para $B' \triangleq in.(\overline{o1}.B' + \overline{o2}.B')$, fica bem clara a diferença fundamental entre estes dois processos (escolha externa *vs.* escolha interna):



Note-se que as árvores de derivação são, geralmente, infinitas, pelo que não as podemos desenhar completamente. Não raro, porém, o grafo de transições correspondente apresenta-se finito [2]. No entanto, podem existir também grafos de transições infinitos. Um exemplo simples é o de um processo que comece por realizar um $in(x)$, para $x \in I\!N$ (porquê?).

5 Semântica

A *semântica* da linguagem de processos que temos vindo a apresentar é definida a dois níveis:

- O nível *arquitectural*, que capta uma noção de similaridade de "configurações de montagem" expressa por uma relação de *congruência estrutural*
- O nível *comportamental* expresso por:
  - *Regras de Reacção*: exprimem o modo como as componentes (de um sistema concorrente) *reagem* (interagem) entre si
  - *Regras de Transição*: generalizam as anteriores exprimindo a evolução do sistema (reacções internas e interacções com o seu "ambiente")

Congruência Estrutural

A sintaxe dos processos força distinções excessivas: por exemplo, $E \mid F$ e $F \mid E$ possuem a *mesma estrutura* e representam, intuitivamente, o mesmo processo: a execução paralela de E e F. Este tipo de constatações conduz à definição de uma relação de equivalência de natureza sintática, que é respeitada pelos operadores da linguagem, a que chamamos *congruência estrutural*.

Definição 2 *A relação de* congruência estrutural *em $\mathbb{P}$ é definida pelo fecho das seguintes clausulas:*

- *Para todo o processo $A(\tilde{x}) \triangleq E_A$, onde $\tilde{x} = \langle x_1, x_2, ..., x_n \rangle$, $A(\tilde{y}) \Leftrightarrow \{\tilde{x}/\tilde{y}\}\, E_A$,* i.e.*processos que podem ser transformados um no outro por* folding/unfolding *das respectivas definições são estruturalmente congruentes.*
- *Conversão-α (substituição de variáveis mudas).*
- *A* composição paralela *e a* escolha *formam, com* $\mathbf{0}$*, monoides abelianos*[3].
- *Para todo $a \notin \mathit{fn}(P)$* $\mathsf{new}\ \{a\}\ (P \mid Q) \Leftrightarrow P \mid \mathsf{new}\ \{a\}\ Q$
- $\mathsf{new}\ \{a\}\ \mathbf{0} \Leftrightarrow \mathbf{0}$

Facilmente se verifica que, como o nome indica, a relação $\Leftrightarrow$ é uma *congruência*.

[2] É o caso do processo B no exemplo. Como exercício compare a árvore e o grafo gerados por $A \triangleq a \cdot A$.
[3] Formulação alternativa: *a ordem das parcelas numa escolha ou dos factores num paralelo é arbitrária.*

Lema 1 *Seja $E \Leftrightarrow F$. Então,*

$$a.E \Leftrightarrow a.F \tag{13}$$
$$E + R \Leftrightarrow F + R \tag{14}$$
$$R + E \Leftrightarrow R + F \tag{15}$$
$$E \mid R \Leftrightarrow F \mid R \tag{16}$$
$$R \mid E \Leftrightarrow R \mid F \tag{17}$$
$$\mathsf{new}\, K\ E \Leftrightarrow \mathsf{new}\, K\ F \tag{18}$$
$$\tag{19}$$

Prova: Exercício.

❑

O seguinte lema, cuja prova é também sugerida ao leitor como exercício, exprime um resultado fundamental sobre a arquitectura dos sistemas concorrentes exprimíveis nesta linguagem.

Lema 2 *Todo o processo é* estruturalmente congruente *a um processo na forma seguinte, dita* concorrente canónica,

$$\mathsf{new}\, K\ (E_1 \mid E_2 \mid ... \mid E_n) \tag{20}$$

onde $K \subseteq L$ e cada E_i é uma escolha não vazia
Prova: Exercício.

❑

Um exemplo importante do recurso a esta forma canónica surge na definição de *operadores de ligação* que abstraem num único conectivo o padrão arquitectural seguido, por exemplo, na especificação de *buffers* com mais do que uma posição. I.e.,

$$E \frown F \triangleq \mathsf{new}\, \{m\}\ (E(..., m, ...) \mid F(..., m, ...)) \tag{21}$$

onde $E(..., x, ...)$ e $F(..., \overline{y}, ...)$, genralizado a um número arbitrário de portas e ligações "cruzadas". É fácil mostrar (mas deixaremos isso em suspenso até à próxima Lição) que o operador satisfaz determinadas propriedades: por exemplo, $\frown$ é associativo, a menos de $\Leftrightarrow$.

Reacção e Transição

A evolução interna de um processo resulta das *interacções* que se estabelecem entre as suas componentes. Esta é captada formalmente por uma *relação de reacção* — o nome, com ressonâncias nas reacções químicas, não é arbitrário. Formalmente,

Definição 3 *A relação de reacção $\longrightarrow$ é definida em $\mathbb{P}$ como o fecho das regras seguintes:*

- *Interacção (ou reacção básica):* $\dfrac{}{a.E \mid \overline{a}.F \longrightarrow E \mid F}\ (react)$
- *Reacção não observável:* $\dfrac{}{\tau.E \longrightarrow E}\ (tau)$
- *Reacções podem ocorrer no contexto de uma composição paralela:* $\dfrac{E \longrightarrow E'}{E \mid F \longrightarrow E' \mid F}\ (par)$
- *ou no interior de uma restrição:* $\dfrac{E \longrightarrow E'}{\mathsf{new}\, C\ E \longrightarrow \mathsf{new}\, C\ E'}\ (local)$

- *Finalmente, a congruência estrutural pode ser usada em qualquer ponto da inferência de reacções:*

  Se $E \Leftrightarrow F$ *e* $E' \Leftrightarrow F'$, *então* $\dfrac{E \longrightarrow E'}{F \longrightarrow F'}\ (struc)$

As regras anteriores, que governam as interacções entre processos, podem ser colocadas num contexto mais amplo. De facto, para captar o significado operacional dos diversos operadores da linguagem, torna-se necessário definir sobre $\mathbb{P}$ um sistema de transição, indexado pelas acções,

$$\{\stackrel{a}{\longrightarrow} \subseteq\ \mathbb{P} \times \mathbb{P} \mid a \in Act\}$$

Claramente, o caso em que $a = \tau$ corresponde a um passo de *reacção*. Vamos descrever esta relação de transição por indução sobre a estrutura (sintática) dos processos. Deste modo obtemos uma descrição

Estrutural *i.e.*, que descreve as transições possíveis para cada "tipo" (ou "forma") de processo (tipo esse ditado pelo conectivo mais externo),

Modular *i.e.*, em que as transições de cada processo são definidas à custa das transições possíveis para os seus sub-processos.

A cada operador sobre processos associa-se, assim, um conjunto de regras de inferência da relação de transição que, no seu todo, constituem a semântica (operacional) da linguagem. O conjunto de regras é *completo* no sentido em que as únicas transições possíveis são as que podem ser inferidas por sua aplicação. Cada regra tem uma conclusão e zero ou mais hipóteses. Numa regra associada a um operador a conclusão será uma transição de um processo composto pelo operador em causa aplicado a determinados argumentos. As hipóteses serão transições de algumas dessas componentes.

Atente-se na diferença entre as regras para os operadores estáticos (em que o operador permanece após a realização da transição, modificando-se apenas as componentes que efectivamente contribuem para a evolução global do processo) e as que concernem aos operadores dinâmicos (onde a estrutura do termo, em particular o conectivo principal, não persiste no tempo).

Definição 4 *O sistema de transição etiquetado subjacente à semântica operacional da linguagem de processos que temos vindo a estudar tem como estados os termos dessa linguagem e como transições todas aquelas que são geradas por aplicação das regras seguintes combinadas com a conversão-*α*:*

$$\frac{}{a.E \stackrel{a}{\longrightarrow} E}\ (prefix) \qquad \frac{\{\tilde{k}/\tilde{x}\}\, E_A \stackrel{a}{\longrightarrow} E'}{A(\tilde{k}) \stackrel{a}{\longrightarrow} E'}\ (ident)\ \ (\text{se } A(\tilde{x}) \triangleq E_A)$$

$$\frac{E \stackrel{a}{\longrightarrow} E'}{E + F \stackrel{a}{\longrightarrow} E'}\ (sum-l) \qquad \frac{F \stackrel{a}{\longrightarrow} F'}{E + F \stackrel{a}{\longrightarrow} F'}\ (sum-r)$$

$$\frac{E \stackrel{a}{\longrightarrow} E'}{E \mid F \stackrel{a}{\longrightarrow} E' \mid F}\ (par-l) \qquad \frac{F \stackrel{a}{\longrightarrow} F'}{E \mid F \stackrel{a}{\longrightarrow} E \mid F'}\ (par-r)$$

$$\frac{E \stackrel{a}{\longrightarrow} E' \quad F \stackrel{\overline{a}}{\longrightarrow} F'}{E \mid F \stackrel{\tau}{\longrightarrow} E' \mid F'}\ (react) \qquad \frac{E \stackrel{a}{\longrightarrow} E'}{\mathsf{new}\,\{k\}\ E \stackrel{a}{\longrightarrow} \mathsf{new}\,\{k\}\ E'}\ (res)\ \ (\text{se } a \notin \{k, \overline{k}\})$$

Os resultados seguintes, cuja demonstração se sugere como exercício, estabelecem a compatibilidade entre a semântica operacional da linguagem de processos, a congruência estrutural e a relação de reacção. Em particular, e como seria de esperar, todas as *reacções* são justificadas (ou explicadas) por transições neste sistema. A partir daqui, e recordando o estudo das bissimulações sobre sistemas de transição arbitrários, conclui-se que a congruência estrutural é uma bissimulação em $\mathbb{P}$ (porquê?).

Lema 3

i) *A congruência estrutural respeita as transições: se* $E \xrightarrow{a} E'$ *e* $E \Leftrightarrow F$ *então existe um processo* F' *tal que* $F \xrightarrow{a} F'$ *e* $E' \Leftrightarrow F'$.

ii) *A relação de reacção é compatível com as transições não observáveis:* $E \xrightarrow{\tau} E''$ *se e só se existe* E' *tal que* $E' \Leftrightarrow E''$ *e* $E \longrightarrow E'$.

Prova: As provas, que são remetidas para o exercício **??**, procedem por indução sobre a profundidade da árvore de inferência.

❑

NOTA: O grafo de transições para um qualquer processo E tem algumas semelhanças com a representação de um autómato não determinista. Um autómato onde o espaço de estados é um subconjunto de expressões em $\mathbb{P}$, o estado inicial coincide com a expressão que define E, o alfabeto é um subconjunto de Act e, por fim, a relação de transição $\delta \subseteq \mathbb{P} \times Act \times \mathbb{P}$ representa a evolução de E passo a passo. Em rigor, essa evolução (e logo δ) traduz o significado que, um tanto intuitivamente, temos vindo a associar aos operadores sobre processos.

Recorde-se, porém, da Lição 2, que os autómatos finitos são demasiado restritivos para servirem de domínio de interpretação para a noção de processo que nos interessa. Em primeiro lugar, e esta é apenas uma questão de "cosmética", o alfabeto de um autómato, *i.e.*, o conjunto de símbolos que este processa para reconhecer frases da linguagem que representa, é aqui entendido no sentido mais vasto de um espaço de interacções ou portas de comunicação. Mais relevante é o facto de os estados finais que, nos autómatos são precisamente os estados em que termina o reconhecimento de frases válidas, deixarem de fazer sentido quando sobre $\mathbb{P}$ queremos definir noções de observação mais elaboradas do que a clássica *linguagem aceite*. Por fim, notemos que a linguagem introduzida nos permite definir processos cujo espaço de estados (dos respectivos grafos de transições) não é finito. Considere-se, por exemplo, a seguinte definição de um contador: $C \triangleq up.(C \mid down.\mathbf{0})$. C é, de resto, equivalente à seguinte expressão onde a não finitude do espaço de estados se expressa por indexação:

$$C_0 \triangleq up.C_1$$
$$C_{i+1} \triangleq up.C_{i+2} + down.C_i$$

Verificação de Transições

A semântica operacional da linguagem permite-nos verificar, por aplicação sucessiva das regras de inferência, a possibilidade de uma determinada transição ocorrer. Na realidade, o facto de o conjunto de regras ser completo, torna possível exprimir a justificação de uma transição na forma de um *diagrama de inferência*, em que cada inferência é etiquetada com o nome da regra aplicada.

Note-se que a árvore de prova cresce da conclusão para as premissas até estas se esgotarem. Tal acontece apenas por aplicação da regra $(pref)$ que, não dependendo de nenhuma hipótese, constitui o único axioma na definição da relação de transição.

Exemplo:

$$\dfrac{\dfrac{}{\overline{m}.B_1 \xrightarrow{\overline{m}} B_1}\,(prefix) \quad \dfrac{\dfrac{}{m.\overline{out}.B_2 \xrightarrow{m} \overline{out}.B_2}\,(prefix)}{B_2 \xrightarrow{m} \overline{out}.B_2}\,(ident)}{\overline{m}.B_1 \mid B_2 \xrightarrow{\tau} B_1 \mid \overline{out}.B_2}\,(react)$$
$$\overline{\overline{m}.B_1 \mid B_2 \xrightarrow{\tau} B_1 \mid \overline{out}.B_2}\,(res)$$

onde $B_1 \triangleq B(in, m)$ e $B_2 \triangleq B(m, out)$.

Porque será que

$$\mathsf{new}\,\{m\}\ (\overline{m}.B_1 \mid B_2) \xrightarrow{\overline{m}} \mathsf{new}\,\{m\}\ (B_1 \mid \overline{out}.B_2)$$

não é uma transição válida?

6 Processos Parametrizados

É agora altura de voltarmos, de forma mais sistemática, ao estudo dos processos com parametrização. Em particular, consideraremos o caso em que as interacções entre processos não se limitam a simples sincronizações, mas podem envolver a troca de informação estruturada. Para simplificar a apresentação, vamos assumir um universo V de valores, um conjunto V_e de expressões sobre V e uma função de avaliação de expressões $Val : V \longleftarrow V_e$. O exemplo simples do *buffer*, que já encontramos atrás,

$$B \triangleq in(x).B'_x$$
$$B'_v \triangleq \overline{out}\langle v\rangle.B$$

serve para ilustrar os ingredientes básicos a considerar na extensão procurada à linguagem base $\mathbb{P}$. Vamos designar esta linguagem estendida por $\mathbb{P}_V$. Assim, temos,

- Duas formas distintas de prefixação: $a(x).E$ e $\overline{a}\langle e\rangle.E$, onde x é uma variável e e uma expressão, representando, respectivamente, acção de entrada e de saída através da porta a.
- Processos parametrizados por um tipo de dados $S \in V$, da forma $A_S(x_1, ..., x_n) \triangleq E_A$, onde cada $x_i \in L$, como habitualmente.

Vamos, ainda, introduzir uma forma de expressão condicional $\mathsf{if}\, b\, \mathsf{then}\, P$ ou, ainda, $\mathsf{if}\, b\, \mathsf{then}\, P_1\, \mathsf{else}\, P_2$. Note-se, porém, que

$$\mathsf{if}\, b\, \mathsf{then}\, P_1\, \mathsf{else}\, P_2 \overset{\text{abv}}{=} (\mathsf{if}\, b\, \mathsf{then}\, P_1) + (\mathsf{if}\, \neg b\, \mathsf{then}\, P_2) \tag{22}$$

Regras Semânticas Adicionais

$$\dfrac{}{a(x).E \xrightarrow{a(v)} \{v/x\}E}\,(prefix_i) \quad \text{para } v \in V$$

$$\dfrac{}{\overline{a}\langle e\rangle.E \xrightarrow{\overline{a}\langle v\rangle} E}\,(prefix_o) \quad \text{para } Val(e) = v$$

$$\frac{E_1 \xrightarrow{a} E'}{\mathsf{if}\, b \,\mathsf{then}\, E_1 \,\mathsf{else}\, E_2 \xrightarrow{a} E'}\,(if_1) \quad \text{para } Val(b) = \mathsf{true}$$

$$\frac{E_2 \xrightarrow{a} E'}{\mathsf{if}\, b \,\mathsf{then}\, E_1 \,\mathsf{else}\, E_2 \xrightarrow{a} E'}\,(if_2) \quad \text{para } Val(b) = \mathsf{false}$$

Esquema de Tradução

Para determinar a semântica deste tipo de processos vamos mostrar como os podemos *traduzir* na linguagem base. Assim, para calcular as transições admissíveis para um processo parametrizado, começamos por o traduzir para $\mathbb{P}$ e, de seguida, determinamos as transições a partir dessa tradução. De facto, a linguagem estendida $\mathbb{P}_V$ pode ser reduzida à linguagem base $\mathbb{P}$ através de um esquema de tradução que a cada nome a de $\mathbb{P}_V$ faz corresponder um conjunto $\{a_v \mid v \in V\}$ de nomes em $\mathbb{P}$. Isto significa que vamos expandir cada porta a num conjunto de portas a_v, uma para cada valor v susceptível de ser comunicado em a. O esquema de tradução é recursivo sobre a estrutura das expressões em $\mathbb{P}_V$. Vamos, pois, definir uma função $\mathcal{T}(\ ) : \mathbb{P} \longleftarrow \mathbb{P}_V$ de acordo com,

$$\mathcal{T}(a(x).E) = \sum_{v \in V} a_v.\mathcal{T}(\{v/x\}E) \tag{23}$$

$$\mathcal{T}(\overline{a}\langle e\rangle.E) = \overline{a}_e.\mathcal{T}(E) \tag{24}$$

$$\mathcal{T}(\sum_{i \in I} E_i) = \sum_{i \in I} \mathcal{T}(E_i) \tag{25}$$

$$\mathcal{T}(E \mid F) = \mathcal{T}(E) \mid \mathcal{T}(F) \tag{26}$$

$$\mathcal{T}(\mathsf{new}\ K\ E) = \mathsf{new}\ \{a_v \mid a \in K, v \in V\}\ \mathcal{T}(E) \tag{27}$$

e, ainda,

$$\mathcal{T}(\mathsf{if}\, b \,\mathsf{then}\, E) = \begin{cases} \mathcal{T}(E) & \text{se } Val(b) = \mathsf{true} \\ \mathbf{0} & \text{se } Val(b) = \mathsf{false} \end{cases} \tag{28}$$

7 Estudo de Casos

Especificação na forma concorrente canónica

O método mais usual para descrever processos com alguma complexidade consiste em recorrer à *forma concorrente canónica*, *i.e.*,

$$P \triangleq \mathsf{new}\ K\ (E_1 \mid E_2 \mid ... \mid E_n)$$

Assim torna-se necessário realizar a

1. Identificação dos sub-processos relevantes;
2. Definição das acções para cada um dos sub-processos (sem grande preocupação com o padrão temporal da sua ocorrência);

3. Desenho do diagrama de sincronização para captar uma visão detalhada das interacções (internas e externas) do sistema;
4. Definição de cada sub-processo por uma expressão em $\mathbb{P}$ (note-se que alguns sub-processos podem ainda ser muito complexos, aplicando-se o método recorrentemente);
5. Exprimir o resultado na forma concorrente canónica e, se necessário, iterar o ciclo de desenvolvimento.

Como exemplo, considere (e analise!) a seguinte especificação de uma máquina de azar onde, após a introdução de uma moeda (sinalizada pela acção m), o cliente pode receber uma quantidade x arbitrária de moedas (acção $\overline{win}\langle x\rangle$) ou ... perder (acção $\overline{loss}$).

$$
\begin{aligned}
IO &\triangleq m.\overline{bank}.(lost.\overline{loss}.IO + rel(x).\overline{win}\langle x\rangle.IO)\\
B(n) &\triangleq bank.\overline{max}\langle n+1\rangle.left(x).B(x)\\
Dc &\triangleq max(z).(\overline{lost}.\overline{left}\langle z\rangle.Dc + \sum_{1\leq x\leq z} \overline{rel}\langle x\rangle.\overline{left}\langle z-x\rangle.Dc)
\end{aligned}
$$

$$
M(n) \triangleq \mathsf{new}\ \{bank, max, left, rel\}\ (IO \mid B(n) \mid Dc)
$$

Construção de padrões sequenciais

Para sistemas mais simples, de comportamento puramente sequencial, adopta-se geralmente um método similar ao usado no construção de autómatos para reconhecimento de linguagens. Assim,

1. Enumeração de todos os estados (*i.e.*, configurações de valores das variáveis) para os quais o sistema pode evoluir;
2. Definição de uma relação que mostre a ordem de evolução do sistema entre os estados considerados;
3. Por fim, definição de expressão em $\mathbb{P}$ que capte as transições identificadas acima.

Vejamos um breve exemplo. Pretende-se realizar um conversor que aceita 3 dígitos binários (começando pelo menos significativo) e detecta se o número é par, ímpar ou nulo. Supõe-se que o processo é activado por uma acção rq (requisição de serviço). Dispõe, ainda, das portas $out0$ e $out1$, para recepção de um 0 ou 1, respectivamente, assim como das portas de resposta $\overline{odd}$, $\overline{even}$, $\overline{zero}$. Identificam-se os seguintes estados:

A O processo aguarda uma requisição de serviço (modelada por rq).

B O processo aguarda entrada do primeiro bit.

C O processo aguarda entrada do segundo bit.

D O processo aguarda entrada do terceiro bit.

Facilmente se definem as transições expressas na expressão seguinte:

$$
\begin{aligned}
A &\triangleq rq.B \\
B &\triangleq out0.C + out1.\overline{odd}.A \\
C &\triangleq out0.D + out1.\overline{even}.A \\
D &\triangleq out0.\overline{zero}.A + out1.\overline{even}.A
\end{aligned}
$$

Exemplo: O *Alternating Bit Protocol*

Como exemplo de uma especificação não trivial de um sistema concorrente, vamos analisar a formalização, na linguagem introduzida, do *Alternating Bit Protocol*, um popular protocolo de comunicações. Recorde-se que um *protocolo* é basicamente um conjunto de regras que governam a interação entre duas ou mais entidades de modo a ser satisfeito um determinado objectivo. Assim, por exemplo, o `HTTPS` (*Secure hypertext transfer protocol*) tem por objectivo a transferência de informação encriptada sobre a Internet. O objectivo do ABP, por seu lado, é garantir a efectividade das comunicações sobre um meio que se assume não ser completamente fiável. Assume-se, em particular, a possibilidade de uma mensagem ser *perdida* ou *replicada* quando em trânsito. A forma como o protocolo controla essas situações é através de um mecanismo de *acknowledgements*. Vejamos esquematicamente o seu funcionamento e o modo como o podemos especificar na linguagem $\mathbb{P}$. Assim,

O processo responsável pelo envio procede do modo seguinte:

- Aceita mensagem para envio
- Envia-a empacotada com o bit b e activa um temporizador
- Quando este atinge "time-out"reenvia b
- Ao receber confirmação b, continua com nova mensagem e $1 - b$
- Ignora qualquer confirmação do tipo $1 - b$

O que nos conduz à seguinte especificação:

$$
\begin{aligned}
Accept(b) &\triangleq accept \cdot Send(b) \\
\\
Send(b) &\triangleq \overline{send_b} \cdot \overline{time} \cdot Sending(b) \\
Sending(b) &\triangleq timeout \cdot Send(b) + ack_b \cdot timeout \cdot Accept(1-b) \\
&\quad + ack_{1-b} \cdot Sending(b)
\end{aligned}
$$

Similarmente, o processo receptor

- Recebe e entrega mensagem
- Envia confirmação com o bit b e activa um temporizador
- Quando atinge "time-out"reenvia b
- Na recepção de uma nova mensagem $1 - b$ entrega-a e prossegue com $1 - b$
- Ignora mensagens etiquetadas com b

Formalmente,

$$Deliver(b) \triangleq deliver \cdot Reply(b)$$

$$\begin{aligned} Reply(b) &\triangleq \overline{reply_b} \cdot \overline{time} \cdot Replying(b) \\ Replying(b) &\triangleq timeout \cdot Reply(b) + trans_{1-b} \cdot timeout \cdot Deliver(1-b) \\ &\quad + trans_b \cdot Replying(b) \end{aligned}$$

Juntando estes processos aos respectivos temporizadores, obtemos

$$Timer \triangleq time \cdot \overline{timeout} \cdot Timer$$

$$Sender(b) \triangleq accept.\mathsf{new}\,\{time, timeout\}\ (Send(b) \mid Timer)$$

$$Receiver(b) \triangleq \mathsf{new}\,\{time, timeout\}\ (Reply(b) \mid Timer)$$

redefinindo $Deliver(b)$:

$$Deliver(b) \triangleq deliver.Receiver(b)$$

O meio de comunicação é especificado de forma a exibir as características de ruído que acima consideramos. Assim,

$$\begin{aligned} Trans(sb) &\triangleq \overline{trans_b} \cdot Trans(s) \\ Trans(s) &\triangleq send_b \cdot Trans(bs) \\ Trans(tbs) &\triangleq \tau \cdot Trans(ts) \\ Trans(tbs) &\triangleq \tau \cdot Trans(tbbs) \end{aligned}$$

e

$$\begin{aligned} Ack(bs) &\triangleq \overline{ack_b} \cdot Ack(s) \\ Ack(s) &\triangleq reply_b \cdot Ack(sb) \\ Ack(sbt) &\triangleq \tau \cdot Ack(st) \\ Ack(sbt) &\triangleq \tau \cdot Ack(sbbt) \end{aligned}$$

Por fim, colocando as três componentes em paralelo e internalizando todas as acções de controlo, obtemos, para $b \in \{0, 1\}$,

$$AB \triangleq \mathsf{new}\,K\ (Sender(1-b) \mid Trans(\epsilon) \mid Ack(\epsilon) \mid Receiver(b))$$

onde $K = \{send_b, ack_b, reply_b, trans_b \mid b \in \{0, 1\}\}$.

A referência para este estudo é a secção 6.3, capítulo 6, de [Mil89]. Apesar de, por enquanto, dispormos apenas de uma notação matemática para descrever processos, o que não nos permite prosseguir a análise deste exemplo com técnicas mais sofisticadas (o que é feito na secção 6.4), podemos, desde já, apreciar a concisão e clareza com que o problema é especificado. Esta é, sem dúvida, uma das vantagens de recorrermos a uma notação sóbria e com uma semântica formal precisa (compare, por exemplo, a especificação feita com a descrição informal que a antecede).

Reflexão

Estudada já a sintaxe e a semântica de uma *linguagem* para descrever o comportamento dos sistemas de transição, e após alguma prática na sua utilização em modelação, novas questões emergem:

Questões

- Em que circunstâncias é possível dizer que dois processos representam (ou modelam) o mesmo comportamento? Que equivalências, que pre-ordens os permitem relacionar?
- Serão as noções de simulação e bissimulação, estudadas anteriormente, úteis nesse caminho?
- Será possível identificar processos que diferem apenas na sua dinâmica interna e, portanto, não externamente observável?
- Como poderemos raciocinar de forma eficaz sobre estas questões? Há um *cálculo* à nossa espera?

Referências

[BB91] J C M Baeten and J A Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2):142–188, 1991.

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

[Hen88] M. C. Hennessy. *Algebraic Theory of Processes*. Series in the Foundations of Computing. MIT Press, 1988.

[Hoa85] C. A. R Hoare. *Communicating Sequential Processes*. Series in Computer Science. Prentice-Hall International, 1985.

[Mil86] R. Milner. Process constructors and interpretations. In H.-J. (IFIP) Kugler, editor, *Information Processing 86*, pages 507–514. Elsevier Science Publishers B. V. (North-Holland), 1986.

[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.

[Rut00] J. Rutten. Universal coalgebra: A theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000. (Revised version of CWI Techn. Rep. CS-R9652, 1996).

[Sch90] D. J. Scholefield. The formal development of real-time systems: a review. YCS-145, University of York, 1990.