Lição 5: O π-*calculus*

Luís Soares Barbosa

Sumário

Esta Lição é uma introdução ao π-calculus, *enquanto extensão do cálculo de processos estudado nas Lições anteriores, aos sistemas móveis. I.e, aos sistemas reactivos cuja estrutura de inter-ligações se altera dinamicamente ao longo da computação. A ênfase é colocada na noção de mobilidade por* passagem-de-nomes *que identificam as ligações por onde os processos interagem, e na sua aplicação à modelação de sistemas.*

1 Introdução

Consideremos o problema de modelar o funcionamento de uma unidade de saúde composta por dois médicos e um recepcionista, da qual se pretendem especificar as seguintes *interacções*:

- O doente é registado à chegada.
- Logo que um médico está livre, o recepcionista envia o próximo doente para consulta.
- A consulta termina com a entrega da receita ao doente.

No cálculo de processos que estudamos até aqui, uma possível solução seria:

$$\begin{aligned} D_{id,sint} &\triangleq \overline{reg}\langle id, sint\rangle \cdot \cdots \\ R &\triangleq reg(n,s) \cdot (livre_1 \cdot \overline{inf}_1\langle n,s\rangle \cdot R + livre_2 \cdot \overline{inf}_2\langle n,s\rangle \cdot R) \\ M_i &\triangleq \overline{livre_i} \cdot inf_i(n,s) \cdot \cdots \end{aligned}$$

em que o processo D modela o comportamento do doente, que tem associado um identificador id e uma descrição de sintomas ($sint$), M_i representa o médico identificado por i e R é o processo que efectua o registo do doente.

Note-se, porém, que nesta linguagem não é possível criar uma interacção directa entre D e M de forma a permitir, por exemplo, a entrega da prescrição ao doente. Toda a interacção entre os processos que modelam as acções do doente e as do médico são estritamente mediadas pelo processo de registo. No entanto, essa interacção seria possível se o processo M pudesse fazer uso do identificador id do doente não apenas como informação 'estática', mas como o identificador de uma nova possibilidade de comunicação, um novo canal, uma *ligação* cujo nome é apenas conhecido em *runtime* mas que, uma vez conhecido, é usado da mesma forma que qualquer outra ligação no modelo (*e.g.*, reg ou $livre_1$). Teríamos, então,

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

$$R \triangleq reg(n,s) \cdot (livre_1 \cdot \overline{inf}_1\langle n,s\rangle \cdot R + livre_2 \cdot \overline{inf}_2\langle n,s\rangle \cdot R)$$

$$\begin{aligned} M_i &\triangleq \overline{livre_i} \cdot inf_i(n,s) \cdot \overline{n}\langle pre(s)\rangle \cdot M_i \\ D_{id,sint} &\triangleq \overline{reg}\langle id, sint\rangle \cdot id(x) \cdot \mathbf{0} \end{aligned}$$

Note-se que D envia o nome id ao processo R mas, posteriormente, utiliza esse mesmo nome para receber alguma informação. Por sua vez R passa essa ligação ao processo que modela o médico escolhido que a vai utilizar para comunicar directamente ao doente a receita $pre(s)$ correspondente aos seus sintomas s.

A ideia subjacente a esta solução — em que ao longo das suas ligações os processos comunicam nomes de outras ligações — sendo extremamente simples, permite resolver de forma elegante diversos problemas de modelação. Consideremos, assim, mais dois requisitos no problema em mãos:

- A solução proposta acima usando o nome do doente como identificador de uma ligação entre este e o médico pode não funcionar se houver na unidade de saúde doentes com o mesmo nome: o risco de uma receita chegar a mão erradas é apreciável. Uma solução será cada doente recorrer a uma ligação *privada*, como se indica a seguir. Note-se como o operador de restrição é usado para declarar um nome *local* que é enviado ao processo R e posteriormente o médico usará na comunicação com o doente.

$$D_{sint} \triangleq \mathsf{new}\, id\, \overline{reg}\langle id, sint\rangle \cdot id(x) \cdot \mathbf{0}$$

- Suponhamos que o número de médicos ao serviço da unidade de saúde não é fixo, mas arbitrário e, portanto, desconhecido do processo R encarregue do registo dos doentes. Este último possui apenas um canal $livre$ no qual recebe, de cada médico em serviço, uma indicação da sua disponibilidade para atender um doente. Essa 'indicação de disponibilidade' é, ela mesma, usada para R passar a informação relevante sobre o doente. Assim,

$$\begin{aligned} M_i &\triangleq \overline{livre}\langle inf_i\rangle \cdot inf_i(n,s) \cdot \overline{n}\langle pre(s)\rangle \cdot M_i \\ R &\triangleq reg(n,s) \cdot livre(x) \cdot \overline{x}\langle n,s\rangle \cdot R \end{aligned}$$

$$D_{sint} \triangleq \mathsf{new}\, id\, \overline{reg}\langle id, sint\rangle \cdot id(x) \cdot \mathbf{0}$$

Este exemplo ilustra os elementos fundamentais do π-*calculus*: o conceito de *nome* enquanto identificador de uma *ligação* e a possibilidade de estes serem comunicados entre processos. O cálculo, em verdade, baseia-se em apenas duas categorias sintácticas: *processos* e *nomes*. Os primeiros correspondem, como até aqui, a padrões de comportamento de sistemas reactivos. Sistemas, contudo, que agora, tendo a possibilidade de comunicar os identificadores de ligações possíveis ou actuais, podem modificar a estrutura das suas ligações ao longo da computação. Os segundos abstraem quer a noção de *acção* do CCS, quer a de *valor* ou *mensagem* susceptível de ser comunicada numa interacção, quer, ainda a de *referência* (ou mesmo *apontador*) usada em programação por objectos e em linguagens imperativas.

Proposto por R. Milner e seus colaboradores em [MPW92], na sequência do desenvolvimento de CCS, o π-*calculus* constitui um modelo matemático e um cálculo para processos cuja topologia de interligações pode ser alterada ao longo da sua execução em resultado de interacções com outros processos. Na práctica este tipo de processos abstraem o comportamento de um tipo de sistemas reactivos que são hoje, e cada vez mais, a norma e não a excepção: os sistemas *móveis*.

Em rigor ao conceber um modelo para estes sistemas poderíamos pensar em processos que fossem eles próprios objecto de comunicação. Diversos fenómenos em computação são captados

por este tipo de modelos: em certos sistemas de objectos, por exemplo, é possível passar procedimentos como argumentos na invocação de métodos. Do mesmo modo é possível migrar código executável entre diferentes lugares numa rede e activa-lo em máquinas hospedeiras.

A noção de *mobilidade* captada pelo π-*calculus* é, contudo, outra. E, num certo sentido, de utilização mais generalizada e conceptualmente mais fundamental: aquilo que se move não é uma entidade computacional mas uma *ligação* que a referencia. Pensemos, por exemplo, nas ligações entre telemóveis ou nas redes de comunicações que intermitentemente se ligam ou desligam. Ou pensemos nas ligações de hipertexto sobre a web. Ou na passagem de argumentos por referência, ou endereço, na invocação de métodos. Assim A noção básica em π-*calculus* é o facto de *uma ligação poder ser transferida* de um processo P para outro processo Q que adquire, consequentemente e a partir desse momento, a capacidade de a utilizar para interagir com terceiros. É o que sucede na transição seguinte em que dois processos interagindo através de um nome comum (m) trocam um outro nome (a) que o processo receptor passa a poder utilizar para, eventualmente, interagir com outros elementos da rede:

$$\overline{m}\langle a\rangle \cdot Server \mid m(cn) \cdot \overline{cn}\langle d\rangle \cdot Client \quad \stackrel{\tau}{\longrightarrow} \quad Server \mid \overline{a}\langle d\rangle \cdot Client$$

Nomes representam *direitos de acesso*: a, acima, representa um acesso a um terceiro processo, que pode, por exemplo, actuar como gestor de um dado recurso. Note-se que se a fôr o único acesso ao gestor desse recurso, o efeito da interacção acima é *mover* (virtualmente) o recurso acedido por a do processo $Server$ para o processo $Client$. Em verdade, a noção de *nome* (e *passagem de nomes*) é pervasiva em computação, significando neste contexto,

$$\boxed{\text{ligação} \equiv \text{acesso} \equiv \text{canal} \equiv \text{apontador} \equiv \text{endereço}}$$

Esta concepção de mobilidade em que o π-*calculus* se baseia é, como se viu, suportada por um esquema de *passagem-de-nomes*: são os nomes, *i.e.*,, os identificadores de ligações reais ou potenciais, que são transferidos entre processos permitindo, desse modo, a reconfiguração da estrutura das ligações. Trata-se de uma concepção, não apenas dotada de uma teoria rica e tratável, em comparação com os chamados cálculos de ordem superior baseados na *passagem-de-processos* referida acima [EN86], mas extremamente expressiva. Em particular, a *passagem-de-processos* pode ser codificada em *passagem-de-nomes* — o aluno interessado é referido para a Parte V de [SW01] onde essa codificação é discutida detalhadamente. Por outro lado, é capaz de captar formas de interacção eventualmente mais subtis: por exemplo a comunicação de um nome entre dois processos pode significar a disponibilização, ou transferência, *apenas parcial* de um acesso a um determinado recurso: diferentes nomes poderão codificar diferentes (tipos de) acesso.

Nota 1 **[Algumas referências para o π-*calculus*]**

Como se referiu já, o π-*calculus* foi originariamente proposto por R. Milner, J. Parrow e D. Walker num trabalho que começou a circular nos finais dos anos 80 e foi publicado em [MPW92]. O primeiro livro de texto, cobrindo quer o cálculo base sem mobilidade quer o π-*calculus*, foi [Mil99], cuja leitura é recomendada. O livro de D. Sangiorgio e D. Walker [SW01] tem um caracter muito mais vasto, mas essencialmente focado nos aspectos teóricos. Um aspecto que, embora não coberto neste curso, poderá interessar o aluno de Ciências da Computação, é a codificação do λ-*calculus* em π-*calculus*, proposta originalmente em [Mil92] e detalhadamente discutida nesse livro.

A verdade é que o π-*calculus* se revelou uma fonte muito fecunda de intuições e resultados essenciais para o entendimento do fenómeno da computação *global* — um dos *grand challenges* propostos para a investigação em Ciências da Computação (*cf.*, `www.nesc.ac.uk/esi/events/Grand_Challenges/`). A partir dele originaram-se diversos cálculos para lidar com os mais diversos aspectos dos sistemas reactivos ubíquos: a assincronia [San01], a segurança [AG99], a distribuição [CG98], entre muitos outros. Surgiram, ainda, diversas propostas de concretização em linguagens de programação (*e.g.*, PICT [PT00], TYCO [VB98], PICCOLA [AN01] ou Cω [BCF02]) e ferramentas de simulação e análise, como o `mwb` [VM94] que adoptaremos neste curso. O endereço

`lamp.epfl.ch/mobility/`

é um excelente ponto de partida para explorar esta galáxia. A partir dele poderá, em particular, ter acesso a alguns textos de introdução ao π-*calculus* que complementam as aulas da disciplina — [Pie96], para motivação e comparação com o λ-*calculus*, [Par01], para uma visão mais técnica.

---

2 Sintaxe

A sintaxe do π-*calculus*, na versão que estudaremos neste curso, é a seguinte:

Definição 1 *Um processo em* π-calculus *é descrito de acordo com o seguinte BNF:*

$$P \ ::= \ \mathbf{0} \mid \alpha \cdot P \mid P + P \mid P \mid P \mid \mathsf{if}\ \phi\ \mathsf{then}\ P \mid \mathsf{new}\ \{x\}\ P \mid A(y_1, \cdots y_n) \tag{1}$$

onde

$$\alpha \ ::= \ a(x) \mid \overline{a}\langle x \rangle \mid \tau \tag{2}$$

e

$$\phi \ ::= \ n = m \mid n \neq m \tag{3}$$

Adimite-se, ainda, a seguinte notação para definição de processos:

$$A(x_1, \cdots x_n) \triangleq P \quad (\textit{onde } i \neq j \Rightarrow x_i \neq x_j) \tag{4}$$

Algumas notas se justificam para acompanhar esta definição.

- Para além do τ, que representa também aqui a acção não observável, cada acção é composta de dois elementos: o nome sobre o qual a interacção ocorre, a que chamamos o *sujeito* da interacção, e o nome efectivamente comunicado, que designamos por *objecto* da interacção.
- Atente-se também na construção condicional, similar à que encontramos na Lição 3 na introdução do cálculo de processos com passagem de valores. Note-se, porém, a forma restrita dos predicados usados aqui, que se justifica pelo facto apenas *nomes* poderem ser trocados numa interacção: a única operação sobre nomes é, como se esperaria, o teste de igualdade.
- Nomes são *capturados* quer numa acção de *input*, $c(a) \cdot P$, quer numa restrição, $\mathsf{new}\ \{a\}\ P$. Em ambos os casos o nome a torna-se mudo em P. Note-se, porém, que até aqui o objecto de uma restrição eram nomes de portas ou canais e estes não podiam ser transmitidos entre processos. No π-*calculus*, contudo, a restrição deixa de ser *estática*: o escopo (ou zona de validade) de um nome local pode variar ao longo da computação [1]. Em $\mathsf{new}\ \{c\}\ (P \mid Q)$ o *nome* c é *interno* a $P \mid Q$. No entanto c é susceptível de ser comunicado por P ou Q: *i.e.*,, transferido para outro processo que poderá usar a ligação restrita representada por c. Por exemplo,

  $$\mathsf{new}\ \{a\}\ (\overline{m}\langle a \rangle \cdot Server \mid Res) \mid m(cn) \cdot \overline{cn}\langle d \rangle \cdot Client \xrightarrow{\tau} \quad \mathsf{new}\ \{a\}\ (Server \mid Res \mid \overline{a}\langle d \rangle \cdot Client)$$

  A possibilidade de nomes locais se moverem é a fonte de boa parte da complexidade e do poder expressivo do cálculo.
- Por fim, convém estar atento a certas convenções popularizadas, mesmo que informalmente, no π-*calculus*. A mais importante é a omissão de **0** no final das expressões. Outra simplificação comum é de $\mathsf{new}\ \{x\}\ P$ em $\mathsf{new}\ x\ P$.

[1] O aluno é convidado a traçar um paralelo entre o operador de restrição em π-*calculus* e a primitiva new em programação por objectos.

Nota 2 **[Substituição e conversão-α]**

Como habitualmente, designamos por $\mathsf{bn}(E)$ o conjunto dos nomes mudos em E, e por $\mathsf{fn}(E)$ o seu complementar em E. Note-se que no π-*calculus* um nome x é tornado mudo num termo E em resultado de uma restrição ($\mathsf{new}\, x\ E$) ou de um prefixo de input ($a(x) \cdot E$).

Um aspecto que merece alguma atenção aqui, como de resto em CCS ou no λ-*calculus*, é a noção de *substituição segura*, *i.e.*,, substituição que não captura (ou emudece) uma variável livre do termo a que se aplica. Por exemplo, a substituição $\{y/x\}$ em

$$a(x) \cdot (\overline{x}\langle b\rangle \mid x(c))$$

é segura, retornando $a(y) \cdot (\overline{y}\langle b\rangle \mid y(c))$. No entanto, a substituição $\{x/b\}$ origina

$$a(x) \cdot (\overline{x}\langle x\rangle \mid x(c))$$

onde um nome previamente livre (b) se tornou mudo. Como é óbvio, a aplicação de substituições deste tipo implica a prévia conversão-α do termo em causa. Por exemplo,

$$\{z/y\}\ (y(x) \mid a(y) \cdot \overline{y}\langle d\rangle \mid \mathsf{new}\,\{z\}\ \overline{y}\langle z\rangle) \quad = \quad z(x) \mid a(y) \cdot \overline{y}\langle d\rangle \mid \mathsf{new}\,\{w\}\ \overline{z}\langle w\rangle$$

Similarmente,

$$\{x/y, b/c\}(a(x) \cdot \mathsf{new}\,\{w\}\ (\overline{x}\langle w\rangle \cdot \overline{c}\langle y\rangle \cdot \mathbf{0})) \tag{5}$$

é (6)

$$a(z) \cdot \mathsf{new}\,\{w\}\ (\overline{z}\langle w\rangle \cdot \overline{b}\langle x\rangle \cdot \mathbf{0}) \tag{7}$$

Note-se, por fim, que um nome pode ser, ao mesmo tempo, *mudo* e *livre* num mesmo termo. É o caso de x em

$$a(x) \cdot P \ \mid\ x(y) \cdot Q \tag{8}$$

Finalmente, e tal como já aconteceu no cálculo discutido nas Lições anteriores, introduz-se uma *congruência estrutural* cujo o papel é 'relaxar' a sintaxe identificando processos que são intuitivamente o mesmo. Esta noção não deve confundir-se com qualquer das equivalências operacionais entre processos cujas identificações são feitas em termos do comportamento exibido relativamente a uma dada semântica operacional. Assim,

Definição 2 *A congruência estrutural, $\equiv$, é a menor relação fechada para as seguintes regras:*

- *conversão-α*
- $(|, \mathbf{0})$ *e* $(+, \mathbf{0})$ *formam monoides abelianos*
- $A(y_1, \cdots, y_n) \equiv P\{y_1, \cdots, y_n/x_1, \cdots, x_n\}$ *se* $A(x_1, \cdots, x_n) \triangleq P$
- *Leis de* extensão de escopo*:*

$$\mathsf{new}\, n\ \mathbf{0} \ \equiv\ \mathbf{0} \tag{9}$$

$$\mathsf{new}\, n\ (P \mid Q) \ \equiv\ P \mid \mathsf{new}\, n\ Q \qquad \textit{se } n \notin \mathsf{fn}(P) \tag{10}$$

$$\mathsf{new}\, n\ (P + Q) \ \equiv\ P + \mathsf{new}\, n\ Q \qquad \textit{se } n \notin \mathsf{fn}(P) \tag{11}$$

$$\mathsf{new}\, n\ \mathsf{new}\, m\ P \ \equiv\ \mathsf{new}\, m\ \mathsf{new}\, n\ P \tag{12}$$

$$\mathsf{new}\, n\ (\mathsf{if}\ x = y\ \mathsf{then}\ P) \ \equiv\ \mathsf{if}\ x = y\ \mathsf{then}\ \mathsf{new}\, n\ P \qquad \textit{se } n \neq x, n \neq y \tag{13}$$

$$\mathsf{new}\, n\ (\mathsf{if}\ x \neq y\ \mathsf{then}\ P) \ \equiv\ \mathsf{if}\ x \neq y\ \mathsf{then}\ \mathsf{new}\, n\ P \qquad \textit{se } n \neq x, n \neq y \tag{14}$$

3 Modelação de Processos Móveis

Controlo de Acessos

Acesso a Rs, tipicamente um processo gestor de um determinado recurso, é guardado por um nome restrito t que funciona como um *trigger*:

$$\mathsf{new}\, t\ (Server \mid t \cdot Rs)$$

Neste esquema, o processo cliente C solicita acesso a Rs via $Server$ que, por sua vez, lhe envia t como *chave de acesso*:

$$\begin{aligned} c(x) \cdot \overline{x} \cdot C \mid \mathsf{new}\, t\ (\overline{c}\langle t\rangle \cdot Server \mid t \cdot Rs) &\xrightarrow{\tau} \mathsf{new}\, t\ (\overline{t} \cdot C \mid Server \mid t \cdot Rs) \\ &\xrightarrow{\tau} \quad \mathsf{new}\, t\ (C \mid Server \mid Rs) \end{aligned}$$

Note-se que o servidor pode enviar t a diversos clientes que *partilharão* o acesso a Rs, em vez de trabalharem sobre cópias independentes deste. Por seu lado Rs pode ser acedido através de diferentes chaves para oferecer serviços distintos:

$$Rs \triangleq t_1 \cdot Rs_1 \mid t_2 \cdot Rs_2 \mid \cdots \mid t_1 \cdot Rs_n$$

Esta situação ilustra a flexibilidade da mobilidade de ligações (e não de processos) adoptada pelo *π-calculus*.

Ligações Privadas

Suponha-se, agora, que diversos clientes estão a aceder a um mesmo servidor mas que este quer enviar dois nomes, n e m, a um cliente determinado garantindo que não vão ser recebidos por qualquer dos outros clientes. Naturalmente, para que tal seja possível, os clientes não deverão partilhar uma mesma ligação ao servidor. Deverá antes este criar uma ligação privada com o cliente em causa, *i.e.*,, declarar um nome local e enviar-lho para uso na comunicação de n e m. Assim,

$$\mathsf{new}\, priv\ (\overline{c}\langle priv\rangle \cdot \overline{priv}\langle n\rangle \cdot \overline{priv}\langle m\rangle \cdot Server)$$

A troca de informação seguindo este tipo de protocolo é tão comum que se costuma registar pelas abreviaturas seguintes o envio e recepção de informação múltipla em canal privado:

$$\begin{aligned} \overline{c}\langle d_1 \cdots d_n\rangle \cdot P &\overset{\text{abv}}{=} \mathsf{new}\, p\ (\overline{c}\langle p\rangle \cdot \overline{p}\langle d_1\rangle \cdot \cdots \cdot \overline{p}\langle d_n\rangle \cdot P) \\ c(x_1 \cdots x_n) \cdot Q &\overset{\text{abv}}{=} c(p) \cdot p(x_1) \cdot \cdots \cdot p(x_n) \cdot Q \end{aligned}$$

Detecção de Terminação

Uma outra situação muito comum é a de um processo, por exemplo C, activar um serviço de outro, por exemplo Rs, e suspender-se até que este sinalize o termo do serviço. De novo, essa sinalização deve ser feita através de uma ligação privada. Assim,

- C envia um nome privado a Rs para este sinalizar terminação,
- a sua continuação fica suspensa da activação um *trigger* por esse nome.

$$\mathsf{new}\, r\ (\overline{c}\langle r\rangle \cdot r \cdot C) \mid c(x) \cdot Rs \xrightarrow{\tau} \mathsf{new}\, r\ (r \cdot C) \mid \{r/x\}Rs$$

Codificação de Condicionais

Vejamos, agora, uma possível codificação de expressões condicionais através da composição paralela com um processo que transmite o valor em teste. A ideia base é que

$$P \triangleq c(x) \cdot (\mathsf{if}\ x = a\ \mathsf{then}\ E + \mathsf{if}\ x = b\ \mathsf{then}\ D)$$

equivale a

$$P' \triangleq c(x) \cdot (\overline{x} \mid (a \cdot E + b \cdot D))$$

desde que não exista nenhum outro processo a interagir com os nomes a e b.

Nota 3

Será possível adoptar o mesmo esquema para condicionais envolvendo testes de desigualdade? Porquê?

Por outro lado, no exercício 10, é discutida a forma como, em certos casos, o próprio uso do operador de escolha não-determinística pode ser substituído por um uso judicioso da composição paralela.

Replicação e Definição Recursiva

Comportamento infinito pode ser expresso quer por definição recursiva, como em CCS, quer por *replicação*. A notação $!P$ representa um número infinito de cópias de P em paralelo, definida pela seguinte regra de congruência estrutural

$$!P \equiv P \mid !P \tag{15}$$

ou, equivalentemente,

$$!P \equiv \underline{fix}\,(X = P \mid X)$$

Por exemplo, o processo

$$CC \triangleq !(in(x) \cdot \overline{out}\langle x \rangle)$$

comporta-se como uma memória sempre disponível para aceitar informação em in mas disponibilizando-a via out por uma ordem arbitrária. De facto,

$$\begin{aligned} CC &\xrightarrow{in(a)} \overline{out}\langle a \rangle \mid !(in(x) \cdot \overline{out}\langle x \rangle) \\ &\xrightarrow{in(b)} \overline{out}\langle a \rangle \mid \overline{out}\langle b \rangle \mid !(in(x) \cdot \overline{out}\langle x \rangle) \end{aligned}$$

O poder expressivo da replicação e da definição recursiva é idêntico. A primeira, porém, é mais facilmente acomodável na teoria (basta adicionar uma cláusula à definição de congruência estrutural enquanto a segunda, como se viu em CCS, requer a introdução de uma categoria sintáctica específica e uma regra na semântica operacional). A definição recursiva explícita, por seu lado, oferece uma sintaxe mais clara na comprensão das expressões, sendo geralmente escolhida na prática para exemplos não triviais. Vejamos como a definição recursiva pode ser expressa em termos de replicação. Seja

$$A_i(\tilde{x}_i) \triangleq E_i$$

uma família finita de definições recursivas usadas num processo Q. Podemos substitui-la por

$$Fam \triangleq !(a_1(\tilde{x}_1) \cdot \hat{E}_1) \mid \cdots \mid !(a_n(\tilde{x}_n) \cdot \hat{E}_n)$$

onde $\hat{P}$ é o processo P após se substituírem todas as invocações $A_i(\tilde{y})$ por um output $\overline{a_i}\langle\tilde{y}\rangle$. Significa isto que o processo cliente Q original e

$$Q' \triangleq \mathsf{new}\,\{a_1, \cdots, a_n\}\; Fam \mid \hat{Q}$$

diferem apenas no facto de o desenrolar de uma definição ser, no último caso, emulado por uma interacção (entre Fam e $\hat{Q}$). Por exemplo, a definição recursiva de um *buffer*

$$Buf(in, out) \triangleq in(x) \cdot \overline{out}\langle x\rangle \cdot Buf\langle in, out\rangle$$

é traduzida em

$$\mathsf{new}\; a\; !(a \cdot in(x) \cdot \overline{out}\langle x\rangle \cdot \overline{a}) \mid \overline{a}$$

Dados como Processos

Neste exemplo vamos começar a discutir um tópico a que voltaremos nas sessões práticas: a representação de estruturas de dados através de processos. Começaremos, aqui, com o exemplo muito simples dos valores Booleanos.

Uma vez que no π-*calculus* apenas existem *nomes* e *processos*, a representação de valores (no seu sentido convencional, *e.g.*, o valor Booleano true) deverá ser feita através de um processo. Por exemplo, os processos

$$True(a, t) \triangleq \overline{a}\langle t\rangle \;\text{ e }\; False(a, f) \triangleq \overline{a}\langle f\rangle$$

emitem, cada um deles, um nome distinto que pode ser usado externamente como se usaria um valor Booleano. Por exemplo, o processo

$$a(x) \cdot (\mathsf{if}\; x = t\; \mathsf{then}\; E + \mathsf{if}\; x = f\; \mathsf{then}\; D)$$

quando composto em paralelo com $True(a, t)$ irá comportar-se como E, enquanto em paralelo com $False(a, f)$ se comportará como D. De certa forma podemos dizer que este processo implementa uma estrutura *case* para controlo de E e D.

Esta representação dos Booleanos é, porém, insatisfatória na medida em que depende de nomes globais (t e f) cujo o papel acaba por ser muito similar ao desempenhado por valores no sentido mais convencional do termo (que não são nada mais, enfim, que nomes globalmente fixos e conhecidos ...). Uma alternativa, mais próxima do espírito do π-*calculus*, é a seguinte

$$\begin{aligned} True(a) &\triangleq a(x, y) \cdot \overline{x} \\ False(a) &\triangleq a(x, y) \cdot \overline{y} \end{aligned}$$

Neste caso o processo $True(a)$ deve ser pensado como uma representação do valor true acessível em a. A sua interacção com outros processos é realizada do modo seguinte: em a recebe do processo 'cliente' dois nomes e escolhe um deles, no caso o primeiro. É a capacidade para fazer essa selecção que define a dinâmica da representação de true, e não o facto de emitir um nome global como na solução anterior.

Com base nesta representação a estrutura *case* discutida acima é definida como

$$Case(a) \triangleq \mathsf{new}\,\{x,y\}\;\overline{a}\langle x,y\rangle \cdot (x \cdot E + y \cdot D)$$

Claramente o paralelo

$$Case\langle a\rangle \mid True\langle a\rangle \tag{16}$$

evolui invisivelmente para E (ver Exercício 14).

Ao contrário do que sucede em Matemática, em que coisas como o número 3 ou o valor $true$ são platonicamente intemporais, a representação deste último em π-*calculus* é, como vimos, consumida após o uso. A situação pode ser remediada por recurso à replicação, definido

$$\begin{aligned} True(a) &\triangleq !\,a(x,y) \cdot \overline{x} \\ False(a) &\triangleq !\,a(x,y) \cdot \overline{y} \end{aligned}$$

Neste caso a expressão (16) evoluiria não para E mas para $E \mid True(a)$ (e Platão ficaria, por certo, satisfeito).

Dispondo de uma representação para os valores Booleanos, é legítimo interrogarmo-nos sobre a possibilidade de definir representações similares para os operadores usuais sobre eles. Consideremos, por exemplo, o caso da negação:

$$Not(a,p) \triangleq !\,a(t,f) \cdot \mathsf{new}\,\{x,y\}\;\overline{p}\langle t,f\rangle \cdot (x \cdot \overline{f} + y \cdot \overline{t})$$

O processo $Not(a,p)$ é paramétrico em dois nomes: a, que representa o canal de acesso do exterior, e p que representa o acesso ao 'argumento'. Assim, $Not(a,p)$ envia ao processo que representa o 'argumento' um par de nomes em que este responde, indicando assim o valor Booleano que está a representar. Como se esperaria, $Not(a,p)$ complementa o valor recebido. Não é difícil perceber que o comportamento de

$$\mathsf{new}\,p\;(Not\langle a,p\rangle \mid True\langle p\rangle)$$

é, a menos de actividade interna, logo não observável, equivalente a $False\langle a\rangle$. O sentido dessa equivalência só poderá ser discutido mais tarde no final desta Lição, altura em que o aluno deverá voltar a esta questão (*cf.*, exercício 21). Esta discussão em torno da representação de dados por processos em π-*calculus* será continuada a partir da Ficha Laboratorial 3.

4 Semântica

Como em CCS, a semântica operacional do π-*calculus* é dada através de um *sistema de transição etiquetado* pelo conjunto de acções permitidas. Formalmente,

Definição 3 *A semântica operacional do* π-calculus *é especificada pelo fecho das seguintes regras de inferência.*

$$\frac{E' \equiv E,\; E \xrightarrow{a} F,\; F \equiv F'}{E' \xrightarrow{a} F'}\;(struct)$$

$$\frac{}{\alpha.E \xrightarrow{\alpha} E}\,(prefix) \qquad \frac{E \xrightarrow{\alpha} E'}{E + F \xrightarrow{\alpha} E'}\,(choice)$$

$$\frac{E \xrightarrow{a} E'}{\mathsf{if}\ x = x\ \mathsf{then}\ E \xrightarrow{a} E'}\,(match) \qquad \frac{E \xrightarrow{a} E', \quad x \neq y}{\mathsf{if}\ x \neq y\ \mathsf{then}\ E \xrightarrow{a} E'}\,(mismatch)$$

$$\frac{E \xrightarrow{a(x)} E', \quad F \xrightarrow{\overline{a}\langle u\rangle} F'}{E \mid F \xrightarrow{\tau} \{u/x\}E' \mid F'}\,(com) \qquad \frac{E \xrightarrow{a} E', \ \mathsf{bn}(a) \cap \mathsf{fn}(F) = \emptyset}{E \mid F \xrightarrow{a} E' \mid F}\,(par)$$

$$\frac{E \xrightarrow{\overline{a}\langle u\rangle} E', \quad a \neq u}{\mathsf{new}\ u\ E \xrightarrow{\overline{a}\mathsf{new}\,\langle u\rangle} E'}\,(open) \qquad \frac{E \xrightarrow{a} E', \quad u \notin a}{\mathsf{new}\ u\ E \xrightarrow{a} \mathsf{new}\ u\ E'}\,(res)$$

Algumas regras merecem certa reflexão. Por exemplo, na regra (par), para a composição paralela, por que razão se exige que F não contenha uma referência muda na acção a? Suponhamos que de $a(x) \cdot P \xrightarrow{a(x)} P$ se infere

$$(a(x) \cdot P) \mid W \xrightarrow{a(x)} P \mid W$$

Num contexto onde $\overline{a}$ seja oferecido tem-se

$$\frac{(a(x) \cdot P) \mid W \xrightarrow{a(x)} P \mid W, \quad \overline{a}\langle z\rangle \cdot Z \xrightarrow{\overline{a}\langle z\rangle} Z}{((a(x) \cdot P) \mid W) \mid \overline{a}\langle z\rangle \cdot Z \xrightarrow{\tau} \{z/x\}(P \mid W) \mid Z}\,(com)$$

Se o nome x for *livre* em W, o processo W seria afectado após a substituição. Como sempre, a solução será renomear x em W via redução-α.

A regra $(prefix)$, por seu lado, é exactamente aquilo que se esperaria. No entanto, qual é o significado de

$$\mathsf{new}\ u\ \overline{a}\langle u\rangle \cdot E$$

em que o nome comunicado é local? Claramente é distinto de $\mathbf{0}$ (o que aconteceria se a restrição afectasse o a):

$$\begin{array}{rl} & a(x) \cdot R \mid (\mathsf{new}\ u\ \overline{a}\langle u\rangle \cdot E) \\ \equiv & \quad \{\ \text{definição de} \equiv\ \} \\ & \mathsf{new}\ u\ (a(x) \cdot R \mid \overline{a}\langle u\rangle \cdot E) \\ \xrightarrow{\tau} & \quad \{\ \text{assumindo que } u \notin \mathsf{fn}(R)\ \} \\ & \mathsf{new}\ u\ (\{u/x\}R \mid E) \end{array}$$

Mas também *não* é análogo ao de

$$\overline{a}\langle u\rangle \cdot E$$

porque

$$
\begin{array}{ll}
 & (a(x) \cdot \mathsf{if}\ x = u\ \mathsf{then}\ F) \mid \overline{a}\langle u\rangle \cdot E \\
\xrightarrow{\tau} & \quad \{\ (com)\ \} \\
 & \mathsf{if}\ u = u\ \mathsf{then}\ F \mid E \\
\equiv & \quad \{\ \text{definição de} \equiv\ \} \\
 & F \mid E
\end{array}
$$

De facto,

$$
\begin{array}{ll}
 & (a(x) \cdot \mathsf{if}\ x = u\ \mathsf{then}\ F) \mid \mathsf{new}\ u\ \overline{a}\langle u\rangle \cdot E \\
\equiv & \quad \{\ \text{definição de} \equiv, \text{conversão-}\alpha\ \} \\
 & \mathsf{new}\ z\ ((a(x) \cdot \mathsf{if}\ x = u\ \mathsf{then}\ F) \mid \overline{a}\langle z\rangle \cdot E) \\
\xrightarrow{\tau} & \quad \{\ (\text{com})\ \} \\
 & \mathsf{new}\ z\ \mathsf{if}\ z = u\ \mathsf{then}\ F \mid E
\end{array}
$$

o que nos conduz a uma forma alternativa de prefixação que abreviaremos por

$$
\overline{a}\mathsf{new}\ \langle u\rangle \stackrel{\text{abv}}{=} \mathsf{new}\ u\ \overline{a}\langle u\rangle \tag{17}
$$

e que designaremos por *output limitado*. A intuição é que um nome local, representado por u, é transmitido ao longo de a, estendendo-se o seu escopo ao processo receptor. Ficamos, assim, no π*-calculus*, com 4 classes de acções:

- a acção interna, representado por τ,
- a acção de output livre, da forma $\overline{a}\langle x\rangle$,
- a acção de output limitado, da forma $\overline{a}\mathsf{new}\ \langle u\rangle$, em rigor uma combinação de output livre e restrição,
- e, por fim, a acção de input, na forma $a(x)$.

Note-se, por fim que a regra $(open)$ é necessária para gerar *outputs limitados*. Como esta forma de acção não é contemplada na regra (com), não pode interagir de forma directa com acções de input. Tais interacções são inferidas por aplicação de $\equiv$ para mover a restrição para o nível mais externo do termo, seguida da regra (res).

Nota 4 **[Semânticas alternativas para o π-*calculus*]**

Uma transição etiquetada por um input

$$
E \xrightarrow{a(x)} F \tag{18}
$$

representa a possibilidade do processo E receber um qualquer valor u através da ligação a e evoluir para $\{u/x\}F$. O nome x, que constitui o objecto da acção em análise, não representa o valor recebido mas sim uma *referência* às posições no termo F onde o valor recebido, quando efectivamente recebido, irá aparecer. Assim, ao considerarmos as transições futuras de F torna-se necessário considerar todas as possíveis instanciações de x. Por analogia com a programação funcional, poderíamos re-escrever a transição (18) como

$$
E \xrightarrow{a} \lambda\, x\ F \tag{19}
$$

evidenciando a dependência funcional da derivação relativamente ao parâmetro de entrada. Esta analogia está base numa apresentação alternativa da semântica do π*-calculus* que recorre a formas do tipo $\lambda\, x\ P$ e $[x]P$ para representar, respectivamente, *abstracções* e *instanciações* de processos de um modo muito intuitivo (ver, por exemplo, [Mil99]).

Note-se, porém, que esta não é a única maneira possível de especificar a semântica de uma acção de input. Um alternativa seria incluir o próprio nome recebido como objecto da acção, como em

$$a(x) \cdot E \xrightarrow{au} \{u/x\}F \tag{20}$$

Atente-se na diferença entre $E \xrightarrow{a(x)} F$ (E recebe um nome que vai substituir x em F) e $E \xrightarrow{ax} F$ (E recebe o nome x e continua como F).

Se se pretender usar este tipo de transições na semântica do π-*calculus*, é necessário alterar a definição 3 para acrescentar um novo axioma:

$$\frac{}{a(x).E \xrightarrow{au} \{u/x\}E}\ (prefix)$$

e modificar a regra (com) para

$$\frac{E \xrightarrow{au} E', \quad F \xrightarrow{\overline{a}\langle u\rangle} F'}{E \mid F \xrightarrow{\tau} E' \mid F'}\ (com)$$

retirando a substituição da conclusão uma vez que esta já foi feita atrás, nomeadamente na inferência da acção de input.

A escolha entre estas duas representações das transições não é completamente indiferente. Em rigor, cada uma delas origina uma semântica para o π-*calculus* ligeiramente diferente. No caso que temos vido a adoptar até aqui a substituição de nomes originada numa interação ocorre *o mais tarde possível*, nomeadamente quando a transição é inferida através da regra (com). No caso discutido nesta nota a substituição é feita, como vimos, *o mais cedo possível*, na inferência da acção. Consequentemente as semânticas resultantes recebem nomes diferentes: em terminologia anglo-saxónica são designadas, respectivamente, por semânticas *late* e *early*. Veremos que esta mesma distinção surge de novo na caracterização das equivalências entre processos π.

---

5 Bissimilaridade

Tal como no cálculo de processos sem mobilidade, a noção de bissimulação é a base para definir equivalências comportamentais. A sua definição deve, contudo, ter em conta a possibilidade de comunicação de nomes ao longo de uma interação. Assim,

Definição 4 *Uma relação binária S entre processos π é uma simulação se, sempre que $\langle E, F\rangle \in S$ e $E \xrightarrow{\alpha} E'$, tal que qualquer nome em* $\mathsf{bn}(\alpha)$ *não ocorre em nenhum dos processos envolvidos, se tiver*

1. *se $\alpha = a(x)$ então $\exists_{F'}\,.\ F \xrightarrow{a(x)} F' \ \wedge\ \forall_u\,.\ \langle\{u/x\}E', \{u/x\}F'\rangle \in S$*
2. *se α não representa um input, então $\exists_{F'}\,.\ F \xrightarrow{\alpha} F' \ \wedge\ \langle E', F'\rangle \in S$*

Repare-se que a primeira cláusula da definição estabalece muito simplesmente que as derivações de uma acção de input (que são sempre funções de nomes para processos) devem estar relacionadas, via S, ponto a ponto. Naturalmente uma simulação simétrica (*i.e.*,, tal que a sua conversa é ainda uma simulação) é uma bissimulação. Tal como em CCS, a relação de bissimilaridade, $\sim$, é definida como a maior bissimulação, ou seja como a união de todas as bissimulações sobre o conjunto dos processos.

Claramente, tem-se

$$a \mid \overline{a} \ \sim\ a \cdot \overline{a} + \overline{a} \cdot a + \tau \tag{21}$$
$$a \mid \overline{c} \ \sim\ a \cdot \overline{c} + \overline{c} \cdot a \tag{22}$$

No entanto, para

$$E \triangleq c(x) \cdot Z + c(x) \cdot \mathbf{0}$$
$$F \triangleq c(x) \cdot Z + c(x) \cdot (\mathsf{if}\ x = u\ \mathsf{then}\ Z)$$

se $Z \not\sim \mathbf{0}$, vem $E \not\sim F$, uma vez que a transição $E \xrightarrow{c(x)} \mathbf{0}$ não pode ser simulada por F para *todas* as substituições (incluindo, naturalmente, a substituição $\{u/x\}$).

Ao contrário do que acontecia em CCS, a relação $\sim$ não é preservada pela acção de input. Por exemplo, apesar da equivalência (22) é fácil de verificar que

$$w(a) \cdot (a \mid \overline{b}) \;\not\sim\; w(a) \cdot (a \cdot \overline{b} + \overline{b} \cdot a) \tag{23}$$

considerando, nomeadamente, a substituição $\{b/a\}$. De facto, $\sim$ não é fechada para substituições arbitrárias (é-o, contudo, para substituições *injectivas* — porquê?), e, por isso, não é preservada pelo input. Ao contrário do que se passa em CCS, $\sim$ não é uma congruência. No entanto,

Lema 1 *A relação de bissimilaridade $\sim$ é uma equivalência preservada por todos os operadores do π-*calculus *excepto input.*

Prova. Exercício 17.

□

Não é, porém, difícil identificar a maior congruência contida em $\sim$:

Definição 5 *Dois processos E e F são estritamente congruentes se*

$$\sigma\, E \;\sim\; \sigma\, F \tag{24}$$

para toda a substituição σ.

Nota 5 **[Bissimulação *late vs* bissimulação *early*]**

A definição de bissimulação dada acima, que requer que as derivações sejam bissimilares após cada input e cada instanciação, é típica daquilo que atrás chamamos a semântica *late* para o π-*calculus*. Pelo contrário, na semântica *early* a transição etiquetada por um input e a correspondente instanciação do argumento são tomadas atomicamente, *i.e.*,, 'acontecem' ao mesmo tempo. Podemos, assim, caracterizar uma noção de bissimulação, dita também ela uma bissimulação *early*, que prescinde da primeira cláusula da definição 4:

Definição 6 *Uma relação binária S entre processos π é uma simulação* early *se, sempre que $\langle E, F\rangle \in S$ e $E \xrightarrow{\alpha} E'$, tal que qualquer nome em $\mathsf{bn}(\alpha)$ não ocorre em nenhum dos processos envolvidos, se tiver*

$$\exists_{F'} .\;\; F \xrightarrow{\alpha} F' \;\wedge\; \langle E', F'\rangle \in S \tag{25}$$

O fecho simétrico de uma simulação *early* designa, como se esperaria, por bissimulação *early*. Correspondentemente o qualificativo *late* é dado às noções de simulação e bissimulação derivadas da definição 4.

Na prática a diferença entre estas duas definições de bissimulação, sendo real, é pouco relavante (mas atenção: confira o exercício 19). O que é importante reter desta discussão é o facto de o π-*calculus* admitir diferentes semânticas e diferentes noções de bissimulação e correspondentes equivalências. É mesmo possível, por exemplo, estudar o efeito de uma bissimulação *early* sobre, por exemplo, uma semântica *late*. O aluno interessado é referido a [SW01] para uma discussão detalhada que está, porém, fora do âmbito deste curso.

Não obstante a discussão na nota anterior, este assunto ficaria incompletamente tratado aqui sem uma referência a uma tentativa de obter uma congruência através da incorporação na própria definição de bissimulação de uma quantificação sobre *todas* as substituições. Essa é a ideia subjacente à chamada bissimulação *aberta* (*open*), introduzida por D. Sangiorgi em [San96], como o fecho simétrico da relação definida por

Definição 7 *Uma relação binária S entre processos π é uma simulação* aberta *se, para toda a substituição σ, sempre que $\langle E, F\rangle \in S$ e $\sigma\, E \xrightarrow{\alpha} E'$, tal que qualquer nome em $\mathsf{bn}(\alpha)$ não ocorre em nenhum dos processos envolvidos, se tiver*

$$\exists_{F'}\ .\ \ \sigma\, F \xrightarrow{\alpha} F' \ \wedge\ \langle E', F'\rangle \in S \tag{26}$$

A diferença entre esta definição e a que emerge da definição 4 é que aqui se requer que as derivações dos processos envolvidos sejam bissimilares sob todas as substituições e não apenas sob aquelas que afectam o argumento da acção de input. A união de todas as bissimulações *abertas* é uma congruência que representaremos por $\sim_o$.

O lema seguinte relaciona as duas congruências apresentadas, cujas diferenças, na prática são pouco relevantes. Do ponto de vista deste curso, porém, a importância da bissimilaridade *aberta* vem do facto de ser essa a noção de equivalência entre processos π implementada no MWB usado nas aulas teórico-práticas. Esta noção de bissimilaridade é a mais estrita que iremos utilizar aqui.

Lema 2 *Se $E \sim_o F$ então E e F são congruentes de acordo com a definição 5. O converso, porém, não é verdadeiro.*

Prova. É trivial verificar que uma bissimulação *aberta* é ainda uma bissimulação no sentido da definição 4 (onde menos substituições são consideradas). Por outro lado, da definição 7 decorre que

$$E\ \sim_o\ F\ \ \Rightarrow\ \ \sigma\, E\ \sim\ \sigma\, F \tag{27}$$

para toda a substituição σ.

Para mostrar que o converso não se verifica basta recorrer ao seguinte contra-exemplo:

$$P\ \triangleq \tau + \tau \cdot \tau$$
$$Q\ \triangleq \tau + \tau \cdot \tau + \tau \cdot (\mathsf{if}\ x = y\ \mathsf{then}\ \tau)$$

Claramente, P e Q são congruentes de acordo com a definição 5, mas $P \not\sim_o Q$ (cf, exercício 20).

□

6 Equivalência Observacional

Tal como em CCS, também aqui vamos estar interessados em equivalências que ignorem a actividade interna dos processos. A definição deste tipo de equivalências, ditas *fracas* ou *observacionais*, segue os passos já conhecidos. Em particular, recorre-se à relação $\stackrel{\epsilon}{\Longrightarrow}$ como abreviatura de $(\xrightarrow{\tau})^*$. Em complemento, define-se

$$\stackrel{\alpha}{\Longrightarrow}\ \stackrel{\text{abv}}{=}\ \stackrel{\epsilon}{\Longrightarrow} \cdot \xrightarrow{\alpha} \cdot \stackrel{\epsilon}{\Longrightarrow} \tag{28}$$

Novamente, em comparação com o que sucedia em CCS, alguma complexidade adicional é introduzida no tratamento das acções de input: quando se simula (observacionalmente) uma transição

$$\xrightarrow{a(x)}$$

por

$$\xrightarrow{\tau} \cdot \xrightarrow{\tau} \cdot \ldots \xrightarrow{\tau} \cdot \xrightarrow{a(x)} \cdot \xrightarrow{\tau} \cdot \ldots \xrightarrow{\tau}$$

é necessário garantir que a substituição, resultante da instanciação de x, é aplicada logo após a transição de input $\xrightarrow{a(x)}$, antes de outras evoluções não observáveis que, muito naturalmente, podem depender do valor que, após recebido, instanciar x. Somos, assim, conduzidos à definição seguinte.

Definição 8 *Uma relação binária S entre processos π é uma simulação fraca se, sempre que $\langle E, F\rangle \in S$ e $E \xrightarrow{\alpha} E'$, tal que qualquer nome em $\mathsf{bn}(\alpha)$ não ocorre em nenhum dos processos envolvidos, se tiver*

1. *se $\alpha = a(x)$ então*

$$\exists_{F''} .\ F \stackrel{\epsilon}{\Longrightarrow} \xrightarrow{a(x)} F'' \ \wedge\ \forall_u \exists_{F'} .\ \{u/x\}\, F'' \stackrel{a(x)}{\Longrightarrow} F' \ \wedge\ \langle \{u/x\}E', F'\rangle \in S \tag{29}$$

2. *se α não representa um input, então $\exists_{F'} .\ F \stackrel{\alpha}{\Longrightarrow} F' \ \wedge\ \langle E', F'\rangle \in S$*

Como habitualmente a equivalência observacional $\approx$ é definida como a maior bissimulação fraca. Como esperaríamos, também, esta equivalência não é uma congruência na medida em que não é preservada nem pelo prefixo no caso de acção de input, nem pela soma. A transformação de $\approx$ numa congruência segue os passos esperados

- fecho para todas as substituições, como no caso da bissimilaridade estrita;
- simulação das transições $\xrightarrow{\tau}$ iniciais através de $\stackrel{\tau}{\Longrightarrow}$ e não de $\stackrel{\epsilon}{\Longrightarrow}$, como vimos para o caso do CCS na Lição 4.

o que nos conduz à definição seguinte:

Definição 9 *Dois processos E e F são observacionalmente congruentes se, para todas as substituições σ, $\sigma\, E \xrightarrow{\alpha} E'$, tal que qualquer nome em $\mathsf{bn}(\alpha)$ não ocorre em nenhum dos processos envolvidos, se tiver*

1. *se $\alpha = a(x)$ então*

$$\exists_{F''} .\ F \stackrel{\epsilon}{\Longrightarrow} \xrightarrow{a(x)} F'' \ \wedge\ \forall_u \exists_{F'} .\ \{u/x\}\, F'' \stackrel{\epsilon}{\Longrightarrow} F' \ \wedge\ \{u/x\}\, E' \ \approx\ F' \tag{30}$$

2. *se α não representa um input, então $\exists_{F'} .\ F \stackrel{\alpha}{\Longrightarrow} F' \ \wedge\ E' \ \approx\ F'$*

e, conversamente, $\sigma\, F \xrightarrow{\alpha} F'$, implicar transições similares a partir de $\sigma\, E$.

Terminamos aqui o estudo de equivalências entre processos no π-*calculus*, apesar da *big picture* ser muito mais vasta (ver, por exemplo, [SW01] ou, para uma introdução mais esquemática, [Par01]).

Referências

[AG99] M. Abadi and A. Gordon. A calculus for cryptographic protocols. *Jour. of Information and Computation*, 143:1–77, 1999.

[AN01] F. Achermann and O. Nierstrasz. Applications = components + scripts: A tour of piccola. In Mehmet Aksit, editor, *Software Architectures and Component Technology*, pages 261–292. Kluwer, 2001.

[BCF02] N. Benton, L. Cardelli, and C. Fournet. Modern Concurrency Abstractions for C#. In Boris Magnusson, editor, *Proc. of ECOOP 2002*, pages 415–440. Springer Lect. Notes Comp. Sci. (2374), 2002.

[CG98] L. Cardelli and A. D. Gordon. Mobile ambients. In N. Nivat, editor, *Proc. First Inter. Conf. on Foundations of Software Science and Computation Structure*, pages 140–155. Springer Lect. Notes Comp. Sci. (1378), 1998.

[EN86] U. Engberg and M. Nielsen. A calculus of communicating systems with label-passing. Report DAIMI iPB-208, Computer Science Department, University of Aarhus, 1986.

[Mil92] R. Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[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.

[Par01] J. Parrow. An introduction to the π-calculus. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*. Elsevier, 2001.

[Pie96] B. Pierce. Foundational calculi for programming languages. In A. B. Tucker, editor, *Handbook of Computer Science and Engineering*. CRC Press, 1996.

[PT00] B. Pierce and D. N. Turner. Pict: A programming language based in the π-calculus. In *Proof, Language and Interaction*. MIT Press, 2000.

[San96] D. Sangiorgi. π-calculus, internal mobility and agent-passing calculi. *Theoretical Computer Science*, 167(1,2):235–274, 1996.

[San01] Davide Sangiorgi. Asynchronous process calculi: the first- and higher-order paradigms. *Theoretical Computer Science*, 253(2):311–350, 2001.

[SW01] D. Sangiorgi and D. Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.

[VB98] V. Vasconcelos and R. Bastos. Core-TyCO, the language definition, version 0.1. DI/FCUL TR 98–3, Department of Informatics, Faculty of Sciences, University of Lisbon, March 1998.

[VM94] B. Victor and F. Moller. The mobility workbench: A tool for the π-calculus. In *Proceedings of CAV'94: Computer Aided Verification*. Springer Lect. Notes Comp. Sci. (818), 1994.

A Exercícios

Exercício 1

Uma vez que o π-*calculus* descreve sistemas cuja arquitectura de interligações se altera dinamicamente, é natural que os correspondentes diagramas de sincronização também evoluam ao longo das transições sofridas pelos processos. Considere os processos seguintes

$$\begin{aligned} P &\triangleq \overline{a}\langle c\rangle \cdot c \cdot E \\ Q &\triangleq a(x) \cdot \overline{b}\langle x\rangle \cdot \mathbf{0} \\ R &\triangleq b(z) \cdot \overline{z} \cdot F \end{aligned}$$

onde se assume que E e F não partilham qualquer nome. Esboce o diagrama de sincronização correspondente a cada uma das possíveis configurações para as quais pode evoluir a sua composição paralela.

Exercício 2

Mostre que o processo

$$\overline{x}\langle y\rangle \cdot \mathbf{0} \mid \mathsf{new}\, y\ (x(z) \cdot E\langle y, z\rangle)$$

pode evoluir invisivelmente para

$$\mathsf{new}\, w\ E\langle w, y\rangle$$

Exercício 3

Especifique um processo que

1. Leia um nome na porta a e o envie duas vezes pela porta b.
2. Leia nomes em duas portas e envie o primeiro nome lido através da segunda porta.
3. Leia nomes em três portas. Se todos forem o mesmo nome nada deverá ser feito. Se dois forem iguais deverá enviar esse nome pela porta restante. Se os três nomes forem distintos deverá repetir as três leituras.

Exercício 4

Discuta a validade das seguintes igualdades formuladas em termos da congruência estrutural:

1. $x(y) \cdot y(z) \ \equiv\ x(z) \cdot y(y)$
2. $\mathsf{if}\ x = x\ \mathsf{then}\ x(y) \ \equiv\ x(y)$
3. $x \mid y \ \equiv\ x \cdot y + y \cdot x$
4. $x(y) \cdot x(z) \mid y(z) \cdot z(y) \ \equiv\ y(y) \cdot y(z) \mid x(z) \cdot x(y)$

Exercício 5

Usando a congruência estrutural, simplifique:

1. $(\mathsf{new}\, x\ \overline{a}\langle x\rangle + \mathbf{0}) \mid \mathbf{0} \mid \mathsf{new}\, x\ \overline{a}\langle x\rangle$
2. $\mathsf{new}\, \{x, y, z\}\ (a(x) \cdot \overline{x}\langle y\rangle \mid \overline{a}\langle z\rangle \mid \mathsf{new}\, z\ \overline{a}\langle z\rangle)$

Exercício 6

Diga em que condições poderá concluir

$$E \equiv \mathsf{new}\, x\ E$$

e efectue a respectiva prova. Determine, a partir daqui, a validade de

$$\mathsf{new}\, x\ \mathsf{new}\, x\ E \equiv \mathsf{new}\, x\ E$$

Exercício 7

Mostre que todas as restrições não guardadas podem ser promoviadas para o nível mais exterior do termo, por exemplo,

$$\mathsf{new}\ \{x\}\ E + F \equiv \mathsf{new}\ \{x\}\ (E + F)$$

Sugestão: Comece por aplicar conversão-α a todos os nomes mudos de forma a torna-los sintaticamente distintos, após o que poderá aplicar as leis de escopo apropriadas. Note que este modo de proceder corresponde à intuição de que é indiferente declarar um nome local ou renomea-lo para um nome sintaticamente distinto de todos os outros: em qualquer caso é inacessível por outro meio.

Exercício 8

Indique todas as transições possíveis para os processos seguintes:

1. $x(y) \cdot y(z) \mid \overline{x}\langle a\rangle$
2. $(x(y) \cdot \overline{y}\langle u\rangle) \mid \mathsf{new}\ u\ \overline{x}\langle u\rangle$
3. $!(\overline{a} \mid a \cdot \overline{x})$
4. $P(x, x)$ onde $P(a, b) \triangleq \overline{a} \mid b$

Exercício 9

Codifique o processo $S \triangleq A \mid B \mid C$ usando replicação para eliminar a invocação recursiva explícita. Considere

$$\begin{aligned} A &\triangleq a(x) \cdot \overline{x}\langle p\rangle \cdot A \\ B &\triangleq b(y) \cdot B \\ C &\triangleq \overline{a}\langle b\rangle \cdot C \end{aligned}$$

Exercício 10

O combinador $+$ é por vezes considerado irrealista do ponto de vista da sua eventual implementação numa linguagem de programação, na medida em que corresponde a uma forma de escolha global síncrona. Por exemplo, em

$$(n \cdot E + \overline{m} \cdot F) \mid (\overline{n} \cdot A + m \cdot B)$$

a escolha envolve as duas componentes em paralelo e deve ser resolvida sincronamente. Se estas estão geograficamente distantes torna-se necessário um protocolo não trivial para assegurar o comportamento esperado. No entanto o uso de $+$ em expressões envolvendo somas *guardadas* pode ser emulado através da composição paralela. Por exemplo,

$$\tau \cdot E\ + \tau \cdot F$$

é equivalente a

$$\mathsf{new}\, o\ (\overline{o} \mid o \cdot E \mid o \cdot F)$$

onde $o \notin \mathsf{fn}(E) \cup \mathsf{fn}(F)$ actua como um 'oráculo'. Estude a generalização deste esquema à emulação de somas guardadas por nomes arbitrários.

Exercício 11

Um *internet deamon* (*e.g.*, `xinetd`) tem como propósito encaminhar as requisições de serviços que recebe para *handlers* apropriados. Suponha que alguém forneceu os seguintes modelos para o *deamon* e para um seu cliente:

$$\begin{aligned} Dm &\triangleq server(service, reply) \cdot \overline{service}\langle reply\rangle \mid\ !(finger(reply) \cdot \overline{reply}\langle user\rangle) \mid\ !(time(reply) \cdot \overline{reply}\langle now\rangle) \\ Cl &\triangleq \mathsf{new}\, p\ (\overline{server}\langle finger, p\rangle \mid\ p(x) \cdot \overline{print}\langle x\rangle) \end{aligned}$$

Explique o seu funcionamento uma vez compostos em paralelo e descreva algumas das suas transições.

Exercício 12

Considere o seguinte processo especificado no π-*calculus*:

$$M \triangleq \mathsf{new}\, c\ (t \cdot \overline{c} \cdot \mathbf{0} \mid !(c \cdot t \cdot \overline{c} \cdot \mathbf{0}))$$

Analise o comportamento deste processo e diga em que é que este se distingue este do comportamento de um outro processo definido recursivamente por $R \triangleq t \cdot R$.

Exercício 13

Considere os seguintes processos especificados no π-*calculus*:

$$\begin{aligned} P &\triangleq !(i(x) \cdot \overline{o}\langle x\rangle \cdot \mathbf{0}) \\ Q &\triangleq \overline{i}\langle u\rangle \cdot \overline{i}\langle v\rangle \cdot Q \end{aligned}$$

Explique de forma sucinta e clara o comportamento do processo $P \mid Q$.

Exercício 14

Reporte-se à discussão sobre a representação dos valores booleanos no π-*calculus* e à definição de

$$Case(a) \triangleq \mathsf{new}\, \{x, y\}\ \overline{a}\langle x, y\rangle \cdot (x \cdot E + y \cdot D)$$

que seleciona E ou D conforme composta com um processo $True(a)$ ou $False(a)$, respectivamente. Seria interessante evoluir a definição $Case(a)$ para um processo que aceitasse como parâmetros os 'argumentos' E e D. Tal não é possível no π-*calculus* de forma directa, uma vez que envolveria uma parametrização por processos e não por *nomes*. Podemos, contudo, proceder da forma seguinte:

- Definir o processo paramétrico em 3 nomes: o próprio acesso, a, e dois outros que farão de *triggers* para processos 'argumento'. Assim,
$$Case(a, t1, t2) \triangleq \mathsf{new}\, \{x, y\}\ \overline{a}\langle x, y\rangle \cdot (x \cdot \overline{t1} + y \cdot \overline{t2})$$
- Preparar versões com *triggers* para cada um dos processos 'argumento', *e.g.*,
$$PcomT(t) \triangleq t \cdot P$$
- Realizar a composição
$$\mathsf{new}\, \{t1, t2\}\ (True\langle a\rangle \mid Case\langle a, t1, t2\rangle \mid EcomT\langle t1\rangle \mid DcomT\langle t2\rangle) \tag{31}$$

Mostre que a expressão (31) evolui invisivelmente para E. Explicite todas as transições que tiver de considerar.

Exercício 15

Forneça uma bissimulação estrita que relacione cada par dos processos seguintes:

1. $\overline{a}\langle x\rangle \cdot (x \mid \overline{u})$ e $\overline{a}\langle x\rangle \cdot (x \cdot \overline{u} \;+\; \overline{u} \cdot x)$
2. $a(x) \cdot (x \mid \overline{u})$ e $a(x) \cdot (x \cdot \overline{u} \;+\; \overline{u} \cdot x \;+\; \mathsf{if}\ x = u\ \mathsf{then}\ \tau)$
3. $P(u)$ e $Q(u,u)$, onde $P(x) \triangleq x \cdot P\langle x\rangle$ e $Q(x,y) \triangleq x \cdot Q\langle x,y\rangle \;+\; y \cdot y \cdot Q\langle x,y\rangle$

Exercício 16

Determine um processo que seja bissimilar, mas não congruente, com

$$a(x) \cdot x \mid \overline{b}\langle y\rangle \cdot \overline{y}$$

e que não contenha o operador de composição paralela.

Exercício 17

Demonstre o lema 1.

Exercício 18

Recorde a discussão nas aulas sobre o facto dos processos $a \mid \overline{b}$ e $a \cdot \overline{b} + \overline{b} \cdot a$ não serem estritamente congruentes. Mostre, agora, que $a \mid \overline{b}$ é estritamente congruente ao processo

$$a \cdot \overline{b} \;+\; \overline{b} \cdot a \;+\; \mathsf{if}\ (a = b)\ \mathsf{then}\ \tau$$

A partir deste resultado conjecture uma *lei da expansão* para o π*-calculus*. Recorde que as leis de expansão têm como objectivo reduzir composições paralelas a somas.

Exercício 19

Considere os seguintes processos

$$E \;\triangleq c(x) \cdot Z + c(x) \cdot \mathbf{0}$$
$$F \;\triangleq c(x) \cdot Z + c(x) \cdot \mathbf{0} + c(x) \cdot (\mathsf{if}\ x = u\ \mathsf{then}\ Z)$$

Mostre que E e F podem ser relacionados por uma bissimulação *early* mas não por uma bissimulação *late*.

Exercício 20

Complete a demonstração da segunda parte do lema 2.

Exercício 21

Reporte-se à discussão sobre a representação dos valores booleanos no π*-calculus*, e as definições dos processos $Not(a,p)$, $True(a)$ e $False(a)$.

1. Mostre que
$$False\langle a\rangle \;\approx\; \mathsf{new}\, p\ (Not\langle a,p\rangle \mid True\langle p\rangle)$$
2. Considere a seguinte representação de um outro conectivo booleano em π*-calculus*:
$$X(a,p,q) \;\triangleq\; !\, a(t,f) \cdot \mathsf{new}\ \{x,y\}\ \overline{p}\langle x,y\rangle \cdot (y \cdot \overline{t} + \mathsf{new}\ \{w,z\}\ (\overline{q}\langle w,z\rangle \cdot (z \cdot \overline{f} + w \cdot \overline{t}))$$
Identifique de que conectivo se trata e mostre que
$$True\langle a\rangle \;\approx\; \mathsf{new}\ \{p,q\}\ (True\langle p\rangle \mid X\langle a,p,q\rangle \mid False\langle q\rangle)$$
