Universidade do Minho

Escola de Engenharia
Departamento de Informática

Métodos de Programação Concorrente

Luís Manuel Soares Barbosa

Monografia
Fevereiro 1998

Nota:

Este texto constitui uma monografia de suporte à disciplina de *Métodos de Progamação IV* da Licenciatura em Matemática e Ciências da Computação. A edição corrente da disciplina não inclui o material coberto no capítulo 4.

Índice

Capítulo 1

Introdução

1.1 Enquadramento e Objectivos

1.1.1 De Onde Partimos?

Uma das abstracções mais populares para descrever o significado de um programa é a noção matemática de *função*, i.é uma regra de cálculo que transforma a informação fornecida em nova informação que constitui a resposta ao problema proposto. Essa resposta é duplamente definida: de cada vez que o cálculo se repete é produzida uma resposta e esta é invariavelmente a mesma.

Esta associação de um significado funcional ao processo computacional é típica não apenas das linguagens de programação funcional (como ML, LISP ou HASKELL), mas igualmente bem sucedida em outros sistemas cujo comportamento se pode descrever em termos de uma *transformação* de valores. Sistemas que se preocupam essencialmente com a *estrutura* e a *transformação* da informação. Por exemplo, podemos modelar uma rotina escrita numa linguagem imperativa por uma função cujo argumento e resultado é de um tipo que modela a noção de memória endereçável (onde reside o estado sobre o qual a rotina actua).

O modelo funcional é muito mais rico do que, à partida, poderíamos suspeitar. De facto, surge dotado de um cálculo clássico, preciso e sóbrio — o λ-*calculus* e sugere, pela sua própria natureza, uma semântica formal para a programação[1], que é estritamente modular[2]. Foi a partir deste modelo que se desenvolveram ramos fundamentais das ciências da computação, nomeadamente a *teoria dos tipos* (com toda a diversidade de conceitos como funções de ordem superior, polimorfismo, subtipos, etc.), a *semântica denotacional* e, por fim, o estudo conjunto da *complexidade* e *análise dos algoritmos*.

Uma generalização simples deste modelo leva-nos à substituição do conceito de função pelo de *relação*. Esta permite a representação do não determinismo inerente a alguns processos computacionais, dado o isomorfismo bem conhecido entre relações de A para B e funções de A para o conjunto potência de B. O modelo está subjacente à programação relacional (ou lógica) de que o PROLOG é um representante quase patriarcal.

Apesar do sucesso destes paradigmas no entendimento da natureza da programação, que permitiu o desenvolvimento notável de linguagens, sistemas e tecnologias, é consensual re-

[1] Programas são funções, ou mais genericamente, homomorfismos, entre conjuntos, eventualmente com alguma estrutura adicional, que representam os tipos da informação envolvida.

[2] Na medida em que a composição de programas é interpretada como composição de funções.

conhecer que a sua aplicabilidade é limitada à esfera da computação *sequencial* — um cantinho, particularmente bem comportado, mas certamente pequeno, do universo computacional. Trata-se de um domínio bem explorado, nos aspectos mais teóricos como nos tecnológicos, em disciplinas anteriores.

A presença de comportamento *concorrente* começa logo por comprometer a composicionalidade de que anteriormente nos vangloriamos. E isto porque programas activados em paralelo podem *interferir* uns com os outros, de uma forma que, de resto, será, em muitos casos, não determinística. Pense-se, por exemplo, na activação paralela dos programas P_1 = `x:=1; x:=x+1` e P_2 = `x:=2`, mesmo tomando as atribuições como acções indivisíveis (i.é, que não podem ser interrompidas).

1.1.2 O Que Nos Ocupa?

A *computação concorrente* constitui um fenómeno que já nos é tecnologicamente familiar. As suas manifestações situam-se quer ao nível básico das arquitecturas paralelas e das redes de processadores, quer ao nível conceptual do desenvolvimento e manutenção de grandes sistemas modulares de software.

Sumariamente, diremos que um sistema exibe, ou realiza, comportamento concorrente sempre que lida *simultaneamente* com diferentes *estímulos*, agregando componentes *autónomas* que *interagem ao longo* (e, não apenas, no ínicio e no fim) da sua execução. Adicionalmente o comportamento será *distribuido* se a informação e a capacidade de processamento (i.é, as entidades e os processos, ou, ainda, os dados e as operações) estiverem disseminados pelas componentes do sistema.

Nesta área se incluem sistemas tão diversos como gestores de redes de serviços (bancários, administrativos, etc.), sistemas operativos distribuidos, controladores de transportes, telefones ou instrumentação industrial, componentes de interacção homem-máquina, gestores distribuidos da produção, entre muitos outros. Todos eles apresentam um conjunto de características e problemas específicos cuja definição, análise e controlo se tornaram preocupações centrais para a Engenharia de Software. Poderíamos citar, a título exemplificativo, o problema da terminação distribuida, com origem na impossibilidade de uma componente poder decidir, por indagação do seu estado interno, se alcançou ou não um ponto terminal. Mas também os que se relacionam com a ocorrência de *deadlock* — ao nível dos recursos ou das comunicações— ou de divergência, ou, ainda, os que se referem à ordenação dos eventos, à exclusão mútua ou à sincronização.

Em todos os casos a descrição deste tipo de sistemas enfatiza os aspectos *comportamentais*, i.é, os padrões de ocorrência dos acontecimentos e a sua evolução no tempo e as suas interacções. Foi, provavelmente, Carl-Adam Petri o primeiro a notar, no início dos anos 60 (em [Pet62]), a necessidade de novos paradigmas para descrever e raciocinar sobre a concorrência. A teoria das redes popularizadas com o seu nome baseia-se precisamente na possibilidade de uma transição num sistema depender de (i.é, interagir com) outra que ocorre noutro sistema que evolui concorrencialmente com o primeiro.

Reparemos que o pequeno exemplo acima referido identifica um (senão **o**) ingrediente fundamental nos sistemas que nos propomos estudar: a *interacção* entre os processos activados concorrentemente[3]. Quando a interacção é de natureza sequencial, encontra representação na estrutura dos termos funcionais e o λ-*calculus* — enquanto teoria das funções recursivas como regras de transformação e do seu comportamento aplicativo — fornece o modelo adequado. No caso geral, contudo, será necessário enriquecer o cálculo, dotando-o de um

[3] De facto dois computadores distintos e activos, sem qualquer ligação entre si, constituem um sistema concorrente que não justificaria as 52 horas lectivas previstas para esta disciplina...

conjunto de identificadores de *portas* usadas para enviar/aceitar valores, eventualmente em simultâneo. No exemplo dado a interação entre os dois programas resumia-se a uma interferência resultante da manipulação de uma variável partilhada. Outras formas de interacção mais complexas ocorrem na prática. Note-se, de passagem, que a semelhança entre a expressão-λ $\lambda x.P$ e a expressão $\alpha_x.P$, que, na linguagem que vamos introduzir significa a prefixação do processo P pela acção de receber um valor na porta α, sugere "λ" como (o) identificador de uma porta de interação do termo, que é única, precisamente na medida em que a avaliação é sequencial.

Outro ingrediente a ter em conta relaciona-se com o lugar que a *observação* tem na análise destes sistemas. Nos sistemas sequenciais estamos sobremaneira preocupados com a estrutura da informação em jogo e as suas transformações. Aqui preocupar-nos-á sobretudo o elemento dual, i.é a observação dos comportamentos dos processos e das suas interacções.

Necessitamos, pois, de uma linguagem (uma teoria e um cálculo) capaz de exprimir interações complexas, os eventos em que, num dado instante, um processo se pode comprometer e os comportamentos que resultam da sua eventual efectivação. O modelo de computação que procuraremos estudar nesta disciplina, e que gostaríamos de ver tão fecundo como as noções de função e relação são para o caso sequencial, será, pois, o conceito de $\boxed{\textit{processo comunicante}}$. A abordagem que faremos da sua caracterização filia-se na família de modelos para a computação concorrente vulgarmente designados por *modelos de termos*, *álgebras de processos* ou *cálculos de processos*. Em particular, estudaremos com detalhe o *Calculus of Communicating Systems* (CCS), introduzido por Robin Milner. Introduziremos, igualmente, uma lógica (o *μ-calculus*) para raciocinar sobre este tipo de sistemas e suas propriedades.

1.1.3 O Que Vamos Abordar?

Neste contexto procuraremos abordar, no decurso da disciplina, as seguintes questões:

1. *Como representar formalmente a noção de processo comunicante e estudar a sua composicionalidade?* Introduziremos para isso uma linguagem muito simples e estudaremos a sua semântica operacional. A noção básica é a de transição etiquetada e, como veremos, o modelo operacional da nossa linguagem emerge como uma generalização da noção de *autómato*, estudada na disciplina de MÉTODOS DE PROGRAMAÇÃO III. Basicamente, consideraremos autómatos não determinísticos, sem estados finais[4] e sobre os quais é possível definir noções de observação mais elaboradas do que a clássica *"linguagem aceite"*. É curioso como, tendo os autómatos constituido um dos primeiros modelos matemáticos da computação, estão na gênese de dois conceitos base das ciências da computação: o de *estrutura de dados* (em que o espaço de estados do autómato é codificado numa álgebra) e o de *processo*, num sentido que procuraremos tornar claro nesta disciplina.

2. *Em que condições podemos dizer que dois processos exibem o mesmo comportamento?* Uma definição de equivalência entre processos é extremamente útil na prática. Suponha-se, por exemplo, que necessitamos de substituir uma determinada componente de um sistema e que a componente original está esgotada ou se tornou muito cara. Será seguro substitui-la por outra? Intuitivamente diremos que a substituição será segura se o comportamento do sistema permanecer essencialmente inalterado. Mas o que é que isso significa? Uma definição formal de equivalência fornece-nos a resposta esperada e estabelece formas de provar que a substituição é segura. Neste sentido iremos

[4] Como nota [Rei88], nos sistemas concorrentes *"initial states do in general not exist and terminal states might denote disaster rather than result"*.

introduzir algumas noções de equivalência entre processos, baseadas no conceito de *bissimulação*, ou seja, na capacidade de dois processos se "imitarem" mutuamente ao longo de toda a sua evolução. Veremos como diversos resultados genéricos de equivalência podem ser captados na forma de equações, o conjunto das quais constitui um poderoso *cálculo* (equacional) que nos ajudará a classificar, simplificar e, em geral, raciocinar sobre processos comunicantes[5].

3. *Como formular as propriedades deste tipo de sistemas (e que propriedades são, de facto, relevantes?) e como raciocinar sobre elas?* Um primeiro ingrediente a ter em conta aqui será a definição de diferentes tipos de *equivalência* entre processos, correspondendo a diferentes capacidades de observação. Procuremos caracterizar uma relação de igualdade que seja congruente com os construtores da linguagem, de modo a tornar possível o raciocínio equacional sobre processos (i.é, a poderosa "substituição de iguais por iguais"). Num segundo momento abordaremos algumas lógicas *modais* (i.é, em que a noção de verdade é relativa ao *modo* como a realidade é percebida, tipicamente, no nosso caso, ao estado de evolução do processo) e *temporais* (i.é, em que a verdade é indexada pelos valores de um domínio que representa o tempo). Procuraremos caracterizar propriedades locais e temporais e introduzir uma breve taxonomia das propriedades típicas dos sistemas concorrentes (cuja implementação corresponde, de forma igualmente típica, a problemas importantes de engenharia).

Questões que ficam de fora incluem tudo o que se prende com a implementação de processos nas plataformas tecnológicas disponíveis — por exemplo, linguagens concorrentes clássicas (como ADA), plataformas *orientadas ao objecto* ou simples programação de sistemas — problema para o qual ainda são pouco conhecidas soluções (formais) satisfatórias. Outro tópico que teria interesse seria o confronto entre a abordagem própria desta disciplina, e outros modelos da computação concorrente, muito diversos em motivação, resultados e prática. Para o aluno mais corajoso, a referência [WN95] fornece um excelente ponto de partida.

1.2 Referências

Os tópicos cobertos neste curso estão amplamente detalhados nas referências seguintes que constituem a *bibliografia recomendada*. Naturalmente haverá algumas (ligeiras) discrepâncias notacionais e outras (menos ligeiras) no que concerne à ordem de introdução dos assuntos e à ênfase colocada em alguns problemas ou conceitos.

[Mil89] Trata-se do livro de Robin Milner, *Communication and Concurrency*, que constitui a referência base sobre a família de cálculos de processos popularizados sob a designação geral de CCS. Os capítulos 2 a 8 cobrem essencialmente a especificação e cálculo de processos. A lógica de Hennessy-Milner, que constitui um caso particular da lógica que introduziremos no capítulo 3 desta monografia, é descrita no capítulo 10.

[Sti95] É uma monografia pedagógica sobre lógicas modais e temporais para processos que, em particular, introduz detalhadamente o *μ-calculus*. Neste curso, porém, sublinharemos mais o poder expressivo da lógica na classificação das propriedades dos processos que as técnicas de verificação dessas propriedades, assunto que ocupa uma parte substancial de [Sti95].

[5]Estas "leis" têm um estatuto similar (e desempenham um papel análogo) ao, por exemplo, das leis da associatividade do produto ou da distributividade da soma sobre o produto no cálculo aritmético.

É recomendável que este texto seja lido tendo à mão algum tipo de ferramenta computacional de suporte à especificação e verificação de sistemas concorrentes. O CWB — *Edinburgh Concurrency Workbench* [MS96] — é uma dessas ferramenta simples, mas razoavelmente versátil, e particularmente adequada aos cálculos de processos discutidos neste texto.

Referências complementares incluem diversas obras, algumas já clássicas, em torno dos cálculos de processos cuja consulta poderá contribuir para uma visão deste domínio mais ampla e crítica do que aquela que este curso pode transmitir. Em particular, [Hoa85] introduz CSP — um cálculo de processos baseado numa noção distinta de sincronização e composição paralela, com assinalável popularidade na prática da Engenharia do Software. Por outro lado, [Hen88] é uma abordagem muito sistemática da semântica dos processos concorrentes, introduzida não apenas ao nível operacional, a que nos limitaremos neste curso, mas também denotacional e axiomático. Note-se que no CWB é possível experimentar as preordens e equivalências introduzidas no livro. Finalmente, [Fen96] constitui uma introdução recente ao uso de CCS na modelação de sistemas e (a uma ligeira generalização da) lógica de Hennessy-Milner para especificação de propriedades. O interesse principal do livro reside nos diversos *case-studies* apresentados. O problema da representação do tempo real, que iremos omitir, é igualmente considerado.

Saliente-se, por fim, que [Koz83] é a referência original do *μ-calculus* modal, em que basearemos a abordagem neste curso das lógicas para processos. Uma visão de síntese sobre lógicas modais e temporais pode ser encontrada em [Sti92]. A utilização destas lógicas na especificação e verificação de sistemas concorrentes, embora numa perspectiva um pouco distinta da que adoptaremos neste curso, é discutida com detalhe em [MP92].

Capítulo 2

Especificação de Processos Comunicantes

2.1 Introdução

Conceitos Básicos

- Sistema *vs.* Ambiente *vs.* Observação
- Sistemas Sequenciais *vs.* Sistemas Concorrentes.

> Um sistema exibe, ou realiza, comportamento *concorrente* sempre que lida *simultaneamente* com diferentes *estímulos*, agregando componentes *autónomas* que *interagem ao longo* (e, não apenas, no ínicio e no fim) da sua execução. Adicionalmente o comportamento será *distribuído* se a informação e a capacidade de processamento (i.é, as entidades e os processos, ou, ainda, os dados e as operações) estiverem disseminados pelas componentes do sistema. Uma terceira classe de sistemas, que não abordaremos neste curso, são ditos de *tempo real* e caracterizam-se por não ser possível, na sua modelação, abstraír de detalhes sobre restricções temporais concretas. Grande parte de todos estes sistemas apresentam requisitos muito apertados de segurança e fiabilidade, o que exige a aplicação de métodos rigorosos (*i.e.*, *matemáticos*) de especificação e desenvolvimento.

- Paradigma de modelação: Função *vs.* Relação *vs.* *Processo*

Acções

As duas noções básicas a que vamos recorrer para modelar sistemas concorrentes são a de *acção* e *processo*. Uma *acção* é um evento que ocorre autonomamente, de forma atómica (*i.e.*, indivisível no tempo), ao nível de abstracção considerado, e que é susceptível de ser observado (por outras componentes do sistema ou pelo seu ambiente). Assim,

Acção = Ponto (porta) de comunicação = Interacção

Apesar de existirem, na prática, diversos regimes de comunicação, veremos que os podemos modelar em termos de uma noção de interacção muito simples, a saber, o *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 a acção não observável representada pela letra τ.

Processos

Um *processo* não é mais do que um padrão de evolução de determinadas capacidades de interacção ao longo do tempo. A isso chamaremos *comportamento.* Assim, não distinguiremos entre processos activos ou passivos: os processos poderão ser *inertes*, *i.e.*, incapazes de comunicar, ou não. Neste último caso exibem sempre um comportamento activo na medida em que participam em comunicações. Por exemplo, na especificação de um protocolo de comunincações, iremos modelar como processos tanto o Emissor de mensagens como o Receptor e, ainda, o meio de comunicação (por exemplo, um canal de capacidade limitada ou uma variável partilhada). Formalmente, um processo será um termo (eventualmente com variáveis) numa linguagem que designaremos por $\mathbb{P}$, da mesma forma que associamos um programa funcional a um termo em notação λ. Antes, porém de introduzir sistematicamente tal linguagem, vamos considerar alguns exemplos.

2.2 Primeiros Exemplos

Buffers Simples

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

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

... com duas saídas: $C \triangleq in.(\overline{o1}.C + \overline{o2}.C)$

... não determinístico: $D \triangleq in.\overline{o1}.D + in.\overline{o2}.D$

Parametrização

Consideremos

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

Podemos encarar $in(x)$ como uma interacção em que um valor é recebido na porta in e associado à variável x. Similarmente, $\overline{out}(x)$ 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):

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

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.*, $len(s) < n-1$) 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

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

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 três operadores fundamentais na linguagem: *composição paralela*, *renomeação* 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 com as renomeações adequadas. 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 concorrencialmente 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,

$$\begin{aligned} B_1 &\triangleq B[\overline{m}(x)/\overline{out}(x)] = in.\overline{m}(x).B_1 \\ B_2 &\triangleq B[m(x)/in(x)] = m.\overline{out}(x).B_2 \\ S &\triangleq (B_1 \mid B_2)\backslash_{\{m\}} \end{aligned}$$

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

$$Bn' \triangleq (B[\overline{m_1}(x)/\overline{out}(x)] \mid B[\overline{m_2}(x)/\overline{out}(x), m_1(x)/in(x)] \mid \ldots \mid B[m_{n-1}(x)/in(x)])\backslash_{\{m\}}$$

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,

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

2.3 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]; uma delas é, de resto, suportada pelo Cwb. 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: $P\backslash_A$, impossibilita a interacção de P através do conjunto de etiquetas em A.

Renomeação: $P[f]$, renomeia as portas de P de acordo com a função f. Note-se que $f : L \longrightarrow L$ respeita a estrutura das etiquetas, i.é, $f(\overline{a}) = \overline{f(a)}$ e estende-se a Act com $f(\tau) = \tau$.

Operadores Dinâmicos

Trata-se de um conjunto de operadores que permitem a definição de 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. Admite uma forma iterada: $\sum_{i\in I} P_i$, onde I é um conjunto de indexação, ou abreviadamente, $\sum \tilde{P}$, sendo $\tilde{P} \stackrel{\text{abv}}{=} \{P_i | \ i \in I\}$ um vector de processos. Claramente, $\sum_{i\in\emptyset} P_i = \mathbf{0}$.

Constantes e Definição Recursiva

Vamos, ainda, considerar a utilização de *constantes*, *i.e.*, processos cujo significado é dado por uma equação definidora da forma $A \triangleq E$. Deste modo é possível, em particular, definir comportamentos recursivos, como em $B \triangleq in.(\overline{o1}.B + \overline{o2}.B)$.

Uma alternativa ao uso de constantes para definir processos recursivos consiste em introduzir explicitamente na linguagem expressões recursivas na forma $\underline{fix}\ (X = P)$. Por exemplo, $\underline{fix}\ (X = a.X)$ representa o processo X tal que $X \triangleq a.X$ (tecnicamente, o menor ponto fixo da equação anterior). A vantagem é separar *recursividade*, que passa a constituir um conectivo ao mesmo nível que os outros, logo mais facilmente manipulável pelo cálculo, de *definição*. Assim a definição recursiva $B \triangleq in.(\overline{o1}.B + \overline{o2}.B)$ é equivalente a (em rigor, é uma abreviatura de) $B \triangleq \underline{fix}\ (X = in.(\overline{o1}.X + \overline{o2}.X))$. No caso geral, $\underline{fix}_j\ (\tilde{X} = \tilde{E})$ representa a componente j da família de processos definida recursivamente por $X_i \triangleq E_i$. Por sua vez $\underline{fix}\ (\tilde{X} = \tilde{E})$ designa a família de componentes $\underline{fix}_j\ (\tilde{X} = \tilde{E})$.

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 $Act = L \cup \{\tau\}$.

Seja, ainda, um conjunto $\mathcal{X}(= \{X, Y, Z, ...\})$ de símbolos de variável e um conjunto $\mathcal{C}(= \{A, B, C, ...\})$ de símbolos de constante sobre o espaço matemático dos processos que queremos descrever.

Estamos, agora, em condições de definir o conjunto $\mathbb{P}$ (das expressões) dos processos como o conjunto dos termos da seguinte linguagem:

$$E \ ::= \ X \mid C \mid a.E \mid \sum_{i \in I} E_i \mid E_0 \,|\, E_1 \mid E\backslash_K \mid E[f] \mid \underline{fix}\ (\tilde{X} = \tilde{E})$$

onde $X \in \mathcal{X}$, $C \in \mathcal{C}$, $a \in Act$, $K \subseteq L$, f é uma função total em L e $\tilde{X}$ e $\tilde{E}$ designam vectores de variáveis e $\mathbb{P}$-expressões, respectivamente. Por convenção daremos maior precedência à prefixação, seguida da restricção e renomeação, relativamente aos restantes operadores.

Note-se que a escolha binária e a inacção são definidas por abreviatura, respectivamente,

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

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

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?).

2.4 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 [1]. Os nomes internos dessas portas anotam-se na região interior da circunferência; nomes externos (resultantes de renomeações) anotam-se paralelamente aos anteriores mas no exterior da circunferência. Portas complementares são ligadas por arcos. Note-se que os contrutores *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, a restricção a K remove as etiquetas externas a e $\overline{a}$ para cada $a \in K$ e, por fim, a renomeação atribui 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

$$P \xrightarrow{\alpha} P'$$

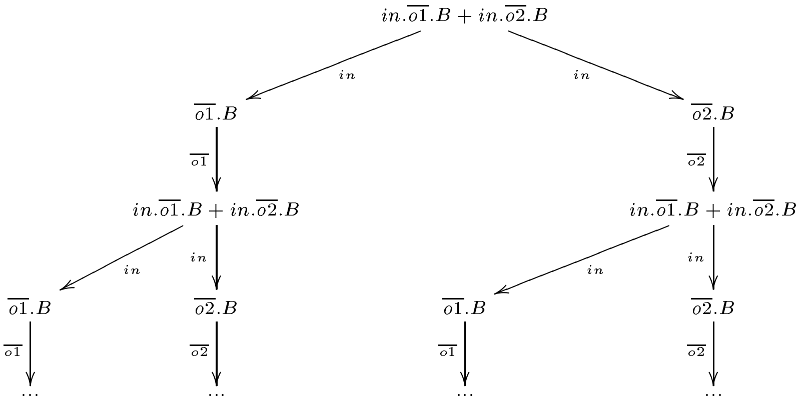
para nos referirmos a uma transição (elementar) de P para P' através da realização de α. Neste caso chamamos ao par (α, P') uma *derivada imediata* de P e a P' uma α*-derivada* de P. Da mesma forma, numa sequência de transições

$$P \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_2} P_2 \xrightarrow{\alpha_3} ... \xrightarrow{\alpha_n} P'$$

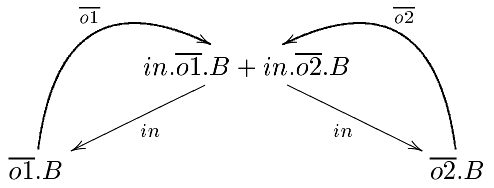
designamos o par $(\alpha_1\alpha_2\alpha_3...\alpha_n, P')$ por *derivada* de P e referimo-nos a P' como uma $\alpha_1\alpha_2\alpha_3...\alpha_n$*-derivada* de P.

Uma maneira de esboçar o comportamento de um processo P consiste em agrupar numa árvore todas as suas derivadas, as derivadas das suas derivadas 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,

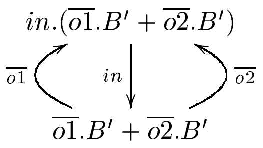
[1] Cf. vários exemplos em [Mil89], cap. 1.



A partir da *árvore de derivação* podemos construir o *Grafo de Transições* colapsando todos os nodos etiquetados pelo 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 $x \in \mathbb{N}$ (porquê?).

2.5 Semântica

Sistemas de Transição

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

[2] É o caso do processo B no exemplo. Como exercício compare a árvore e o grafo gerados por $A \triangleq \alpha.A$.

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. É precisamente nesta noção de sistema de transição que vamos basear a semântica da linguagem $\mathbb{P}$.

Note-se, porém, que os autómatos finitos estudados em Teoria das Linguagens (abordada em MP III) 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 são finitos. 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:

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

Semântica Operacional

A definição da semântica da linguagem $\mathbb{P}$ pode ser feita, essencialmente, por dois modos:

Denotacional

Define-se um espaço de estruturas formais bem caracterizadas (por exemplo, autómatos ou, mais apropriadamente, *sistemas de transição* que são autómatos sem restricções de cardinalidade no espaço de estados e sem identificação de estados finais) e um *homomorfismo* de interpretação nesse espaço dos termos em $\mathbb{P}$. A condição de homomorfismo garante, como é conhecido da Álgebra Universal, que a interpretação de um termo composto (*e.g.*, $P \mid Q$) corresponde à composição das interpretações dos factores (*e.g.*, à composição paralela definida sobre *sistemas de transição* das interpretações de P e Q).

Operacional

Neste caso definimos sobre a totalidade de $\mathbb{P}$ uma estrutura de transição

$$\longrightarrow \subseteq \mathbb{P} \times Act \times \mathbb{P}$$

que capta o significado operacional dos operadores. Em rigor, introduzimos uma família de relações

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

que é equivalente a $\longrightarrow$ pelo isomorfismo $\mathcal{P}(\mathbb{P} \times \mathbb{P})^{Act} \simeq \mathcal{P}((\mathbb{P} \times \mathbb{P}) \times Act) \simeq \mathcal{P}(\mathbb{P} \times Act \times \mathbb{P})$. A sua definição será

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

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

Regras Semânticas

A cada operador sobre processos associamos, pois, 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).

$$\frac{}{a.E \xrightarrow{a} E}\,(P) \qquad\qquad \frac{E_j \xrightarrow{a} E'_j}{\sum_{i\in I} E_i \xrightarrow{a} E'_j}\,(S) \quad \text{para } j \in I$$

$$\frac{E \xrightarrow{a} E'}{E \mid F \xrightarrow{a} E' \mid F}\,(C_1) \qquad\qquad \frac{F \xrightarrow{a} F'}{E \mid F \xrightarrow{a} E \mid F'}\,(C_2)$$

$$\frac{E \xrightarrow{a} E' \quad F \xrightarrow{\overline{a}} F'}{E \mid F \xrightarrow{\tau} E' \mid F'}\,(C_3)$$

$$\frac{E \xrightarrow{a} E'}{E[f] \xrightarrow{f(a)} E'[f]}\,(F) \qquad\qquad \frac{E \xrightarrow{a} E'}{E\backslash_A \xrightarrow{a} E'\backslash_A}\,(R) \quad \text{se } a, \overline{a} \notin A$$

$$\frac{E \xrightarrow{a} E'}{P \xrightarrow{a} E'}\,(D) \quad \text{se } P \triangleq E$$

$$\frac{E[\underline{fix}\ (X = E)/X] \xrightarrow{a} E'}{\underline{fix}\ (X = E) \xrightarrow{a} E'}\,(Rec) \qquad\qquad \frac{E_j[\underline{fix}\ (\tilde{X} = \tilde{E})/\tilde{X}] \xrightarrow{a} E'}{\underline{fix}_j\ (\tilde{X} = \tilde{E}) \xrightarrow{a} E'}\,(Rec_j)$$

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 (P) que, não dependendo de nenhuma hipótese, constitui o único axioma na definição da relação de transição.

Exemplo:

$$\dfrac{\dfrac{\dfrac{}{\overline{m}.B_1 \xrightarrow{\overline{m}} B_1}\,(P) \qquad \dfrac{\dfrac{}{m.\overline{out}.B_2 \xrightarrow{m} \overline{out}.B_2}\,(P)}{B_2 \xrightarrow{m} \overline{out}.B_2}\,(D)}{\overline{m}.B_1 \mid B_2 \xrightarrow{\tau} B_1 \mid \overline{out}.B_2}\,(C_3)}{(\overline{m}.B_1 \mid B_2)\backslash_{\{m\}} \xrightarrow{\tau} (B_1 \mid \overline{out}.B_2)\backslash_{\{m\}}}\,(R)$$

onde $B_1 \triangleq in.\overline{m}.B_1$ e $B_2 \triangleq m.\overline{out}.B_2$.

Porque será que

$$(\overline{m}.B_1 \mid B_2)\backslash_{\{m\}} \xrightarrow{\overline{m}} (B_1 \mid \overline{out}.B_2)\backslash_{\{m\}}$$

não é uma transição válida?

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 derivadas) 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 (que mais tarde justificaremos) dependerem de condições sobre as espécies dos processos envolvidos. Por exemplo, a lei

$$(P \mid Q)\backslash_{\{a\}} \;=\; P\backslash_{\{a\}} \mid Q$$

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

A espécie mínima de um processo P — *i.e.*, $\bigcap\{K \subseteq L \mid\, P : K\}$ — 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.

Mínima, em muitos casos, e, sempre fácil de calcular, é a *espécie sintática* que se representa por $\mathbb{L}_s(P)$ e se define de acordo com,

$$\begin{aligned}
\mathbb{L}_s(a.P) &= \{a\} \cup \mathbb{L}_s(P) \\
\mathbb{L}_s(\tau.P) &= \mathbb{L}_s(P) \\
\mathbb{L}_s(\sum_{i \in I} P_i) &= \bigcup_{i \in I} \mathbb{L}_s(P_i) \\
\mathbb{L}_s(P \mid Q) &= \mathbb{L}_s(P) \cup \mathbb{L}_s(Q) \\
\mathbb{L}_s(P\backslash_K) &= \mathbb{L}_s(P) - (K \cup \overline{K}) \\
\mathbb{L}_s(P[f]) &= f(\mathbb{L}_s(P)) \quad i.e., \{f(a) | \; a \in \mathbb{L}_s(P)\}
\end{aligned}$$

e, para cada $P \triangleq E$, $\mathbb{L}_s(P) \subseteq \mathbb{L}_s(E)$. Note-se, no entanto, que nem sempre a espécie sintática é mínima. Por exemplo, o processo $(a.b.c.0)\backslash_a$ não tem transições, logo a sua espécie mínima é $\emptyset$, embora $\mathbb{L}_s((a.b.c.0)\backslash_a) = \{b, c\}$.

2.6 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 [3] de valores, um conjunto V_e de expressões sobre V e uma função de avaliação de expressões $Val : V_e \longrightarrow V$. O exemplo simples do *buffer*, que já encontramos atrás,

$$\begin{aligned}
B &\triangleq in(x).B'(x) \\
B'(v) &\triangleq \overline{out}(v).B
\end{aligned}$$

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 $\overline{a}(e).E$, onde x é uma variável e e uma expressão, representando, respectivamente, acção de entrada e de saída através da porta a.
- Processos parametrizados, da forma $P(X_1, ..., X_n) \triangleq E$, onde E contém apenas como variáveis livres X_1 a X_n. Recorde-se que em $\mathbb{P}$, quando queríamos definir um conjunto de processos similares, recorríamos não a parametrização mas à construção explícita de uma família indexada de processos (*e.g.*, o exemplo do Contador discutido em aulas anteriores).

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

$$\text{if } b \text{ then } P_1 \text{ else } P_2 \stackrel{\text{abv}}{=} (\text{ if } b \text{ then } P_1) + (\text{ if } \neg b \text{ then } P_2)$$

[3] Esta hipótese não implica perda de generalidade. De facto, V pode ser pensado como um universo de tipos de dados similar aos encontrados nas linguagens de programação ou aos tipos abstractos introduzidos em Especificação Formal (*e.g.*, em MPF I e II).

Regras Semânticas Adicionais

$$\frac{}{a(x).E \xrightarrow{a(v)} E[v/x]}\ (P_i) \qquad \text{para } v \in V$$

$$\frac{}{\overline{a}(e).E \xrightarrow{\overline{a}(v)} E}\ (P_o) \qquad \text{para } Val(e) = v$$

$$\frac{E_1 \xrightarrow{a} E'}{\text{if } b \text{ then } E_1 \text{ else } E_2 \xrightarrow{a} E'}\ (If_1) \qquad \text{para } Val(b) = \top$$

$$\frac{E_2 \xrightarrow{a} E'}{\text{if } b \text{ then } E_1 \text{ else } E_2 \xrightarrow{a} E'}\ (If_2) \qquad \text{para } Val(b) = \bot$$

Esquema de Tradução

Para determinar a semântica deste tipo de processos vamos mostrar como oa 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 etiqueta a de $\mathbb{P}_V$ faz corresponder um conjunto $\{a_v |\ v \in V\}$ de etiquetas 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 $\mathcal{T}(\ ) : \mathbb{P}_V \longrightarrow \mathbb{P}$ de acordo com,

$$\begin{aligned}
\mathcal{T}(X &= X) \\
\mathcal{T}(E(X_1, ..., X_n)) &= E_{X_1,...,X_n} \\
\mathcal{T}(a(x).E) &= \sum_{v \in V} a_v.\mathcal{T}(E[v/x]) \\
\mathcal{T}(\overline{a}(e).E) &= \overline{a}_e.\mathcal{T}(E) \\
\mathcal{T}(\sum_{i \in I} E_i) &= \sum_{i \in I} \mathcal{T}(E_i) \\
\mathcal{T}(E \mid F) &= \mathcal{T}(E) \mid \mathcal{T}(F) \\
\mathcal{T}(E \backslash_K) &= \mathcal{T}(E) \backslash_{\{a_v|\ a \in K, v \in V\}} \\
\mathcal{T}(E[f]) &= \mathcal{T}(E)[\mathcal{T}(f)] \quad \text{onde } \mathcal{T}(f)(a_v) = f(a)_v
\end{aligned}$$

e, ainda,

$$\mathcal{T}(\text{ if } b \text{ then } E) = \begin{cases} \mathcal{T}(E) & \text{se } Val(b) = tt \\ \mathbf{0} & \text{se } Val(b) = f\!f \end{cases}$$

Finalmente, a definição de uma constante $\tilde{P}(\tilde{X}) \triangleq \tilde{E}$ é traduzida num conjunto indexado de definições, *i.e.*, $\{P_{\tilde{v}} \triangleq \mathcal{T}(\tilde{E})[\tilde{v}/\tilde{X}]|\ \tilde{v} \in V \times ... \times V\}$.

2.7 Estudo de Casos

Como exemplo de uma especificação razoavelmente complexa de um sistema concorrente, vamos analisar a formalização, na linguagem introduzida, do *Alternating Bit Protocol*, um popular protocolo de comunicações. 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).

NOTA: No final desta parte do programa convirá deixar uma nota breve sobre os dois *métodos* que temos vindo a seguir para especificar porcessos.

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

   $$P \triangleq (E_1[f_1] \mid E_2[f_2] \mid ... \mid E_n[f_n])\backslash_K$$

   Assim torna-se necessário realizar a

   (a) Identificação dos sub-processos relevantes;
   (b) Definição das acções para cada um dos sub-processos (sem grande preocupação com o padrão temporal da sua ocorrência);
   (c) Desenho do diagrama de sincronização para captar uma visão detalhada das interacções (internas e externas) do sistema;
   (d) 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);
   (e) Exprimir o resultado na forma $P \triangleq (E_1[f_1] \mid E_2[f_2] \mid ... \mid E_n[f_n])\backslash_K$ e, se necessário, iterar o ciclo de desenvolvimento.

2. Para sistemas mais simples, de comportamento puramente sequencial, podemos:

   (a) Fazer uma lista de todos os estados (*i.e.*, configurações de valores das variáveis) para os quais o sistema pode evoluir;
   (b) Definir uma relação que mostre a ordem de evolução do sistema entre os estados considerados;
   (c) Definir uma 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:

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

Exercícios

Exercício I.1

Considere a seguinte descrição de um *buffer* de duas posições com sinalizadores. Note que o processo é construido a partir de um *buffer* simples (de 1 posição) que lida igualmente com sinalizadores. Em particular, sinaliza (em $\overline{r}$) a recepção de uma mensagem. Do mesmo modo, aguarda a confirmação (t) de que a transmissão por si realizada se completou com sucesso.

$$\begin{aligned} Bs &= (B[\overline{mo}/\overline{out}, mi/t] \mid B[mo/in, \overline{mi}/\overline{r}])\backslash_{\{mo,mi\}} \\ B &= in.\overline{out}.t.\overline{r}.B \end{aligned}$$

1. Esboce o diagrama de sincronização do processo Bs e calcule $\mathbb{L}_s(Bs)$.
2. Compare o comportamento de B e Bs (em particular, verifique se Bs se comporta, de facto, como um *buffer* de duas posições). Fundamente a sua resposta construindo e comparando os respectivos grafos de transições.
3. Procure uma solução para o problema detectado (se ele existir!) e trace de novo o grafo de transições relativo a essa solução.
4. Explique como a especificação inicial (ou a sua nova solução) pode ser adaptada para descrever *buffers* com um número qualquer (mas previamente fixo) de posições.
5. Repita a alínea anterior para *buffers* com um número arbitrário (não conhecido à partida) de posições.

Exercício I.2

Considere a seguinte descrição de um *buffer* de uma posição, bi-direcional, *i.e.*, capaz de transmitir um número arbitrário de mensagens em qualquer direcção.

$$BT \triangleq in_1(x).\overline{out_1}(x).BT + in_2(x).\overline{out_2}(x).BT$$

1. Construa um *buffer* de duas posições, igualmente bi-direcional, por composição paralela de duas réplicas do processo BT.
2. Esboce o respectivo diagrama de sincronização.
3. Calcule o seu grafo de transições.

Exercício I.3

Considere a especificação seguinte de um sistema de controlo de um cruzamento entre uma estrada e uma via férrea ([Sti95]). As acções car e $train$ representam a aproximação do cruzamento por um automóvel ou um comboio, respectivamente. Por seu lado, up e dw representam a abertura e o fecho da cancela sobre a estrada, enquanto $green$ e red modelam a recepção de um sinal de avanço ou paragem pelo comboio. Finalmente, as acções $\overline{ccross}$ e $\overline{tcross}$ traduzem, respectivamente, a travessia efectiva do cruzamento por um automóvel ou um comboio.

$$\begin{aligned} Road &\triangleq car.up.\overline{ccross}.\overline{dw}.Road \\ Rail &\triangleq train.green.\overline{tcross}.\overline{red}.Rail \\ Signal &\triangleq \overline{green}.red.Signal + \overline{up}.dw.Signal \end{aligned}$$

$$C \triangleq (Road \mid Rail \mid Signal)\backslash_{\{green,red,up,dw\}}$$

1. Explique o comportamento deste processo e esboce o respectivo diagrama de sincronização.
2. Calcule o grafo de transições correspondente ao processo C

Exercício I.4

Considere a seguinte especificação de uma máquina de venda automática de chocolates, que fornece dois tipos de chocolates (grande e pequeno) ao preço de 100 e 200 escudos, respectivamente ([Mil89]).

$$VM \triangleq i200.grd.recolhe.VM + i100.peq.recolhe.VM$$

Assim, por exemplo, para comprar um chocolate pequeno, o utilizador é suposto começar por introduzir uma moeda de 100, accionar o botão *peq* e recolher a sua compra na saída da máquina.

1. Modifique VT de modo a que, após ter inserido 100, seja possível ou recolher um chocolate pequeno ou voltar a inserir outra moeda de 100 e recolher um chocolate grande.
2. Modifique VT de modo a que, após ter inserido 200, seja possível recolher um chocolate grande ou dois pequenos.

Exercício I.5

No contexto do exercício anterior, especifique uma máquina de venda automática de chocolates mais sofisticada, que admite interacções arbitrárias em qualquer uma das suas 5 portas, desde que sejam respeitadas as seguintes restricções:

- Na gaveta da máquina apenas pode estar, no máximo, um chocolate por recolher;
- Não é possível inserir mais de 400 escudos sem antes escolher e realizar uma compra;
- A máquina "e “honesta”: nunca perde nem ganha dinheiro numa transacção.

Exercício I.6

Um n-*trigger*, para $n > 1$, é um dispositivo, tipicamente utilizado em votações electrónicas, com n portas de entrada, a_1 a a_n, e uma porta de saída $\overline{s}$. Logo que detecte ter recebido um sinal em mais de metade das portas de entrada, o n-*trigger* emite um sinal em $\overline{s}$ (que, eventualmente despoletará outro processo), e termina. Cada porta a_i recebe apenas um sinal e assume-se que estes podem chegar às diferentes portas de entrada por qualquer ordem.

1. Especifique um 3-*trigger* (*i.e.*, um *trigger* com 3 portas de entrada).
2. Especifique um n-*trigger*, para n arbitrário.

Exercício I.7

Seja $A \triangleq a.A$ e $B \triangleq b.B$. Calcule as derivadas imediatas dos processos seguintes:

1. $A + B$
2. $A + B[\overline{a}/b]$
3. $A \mid B$
4. $A \mid B[\overline{a}/b]$
5. $(A \mid B)[\overline{a}/b]$
6. $(A \mid B[\overline{a}/b])\backslash_{\{a\}}$

Exercício I.8

Seja $A \triangleq a.b.A + c.d.A$. Construa os grafos de transições correspondentes aos processos seguintes:

1. A
2. $(A \mid A[\overline{a}/a])\backslash_{\{a\}}$
3. $(A \mid A[\overline{a}/a, \overline{c}/c])\backslash_{\{a,c\}}$

Exercício I.9

Calcule a espécie sintática dos processos considerados no exercício anterior.

Exercício I.10

Qual é o conjunto de derivadas do processo $T \triangleq a.(b.\mathbf{0} \mid T)$?

Exercício I.11

Construa os grafos de transições correspondentes aos processos seguintes, assumindo que a variável x toma valores no conjunto $\{1, 2, 3\}$.

1. $a(x).\overline{b}(x).\mathbf{0}$
2. $(\overline{a}(2).\mathbf{0} \mid a(x).\overline{b}(x).\mathbf{0})\backslash_{\{a\}}$
3. $a(x).(\text{ if } x = 2 \text{ then } \overline{b}(x).\mathbf{0} \text{ else } \overline{c}(x).\mathbf{0})$
4. $(\overline{a}(2).\mathbf{0} \mid \text{ if } x = 2 \text{ then } \overline{b}(x).\mathbf{0} \text{ else } \overline{c}(x).\mathbf{0})\backslash_{\{a\}}$

Exercício I.12

Compare os grafos de transições dos processos seguintes, e discuta até que ponto será razoável considerar equivalentes os comportamentos exibidos.

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

Exercício I.13

Considere os seguintes processos:

$$P \triangleq a.\overline{b}.c.P + \overline{b}.c.a.P + c.a.\overline{b}.P$$
$$Q \triangleq d.\overline{c}.b.Q + \overline{c}.b.d.Q + b.d.\overline{c}.Q$$

$$R \triangleq (P \mid Q)\backslash_{\{b,c\}}$$

e verifique as transições

1. $R \xrightarrow{d} (P \mid \overline{c}.b.Q)\backslash_{\{b,c\}}$
2. $R \xrightarrow{\tau} (a.\overline{b}.P \mid b.d.Q)\backslash_{\{b,c\}}$
3. $(c.P \mid \overline{c}.Q)\backslash_{\{b,c\}} \xrightarrow{\tau} R$

Exercício I.14

Mostre que $(A' \mid B)\backslash_c \xrightarrow{\tau} (A \mid B')\backslash_c$ se verifica, assumindo que $A \triangleq a.A'$, $B \triangleq c.B'$, $A' \triangleq \overline{c}.A$ e $B' \triangleq \overline{b}.B$.

Exercício I.15

Complete as transições seguintes e verifique-as:

1. $P \xrightarrow{a} \cdots$, sendo $P \triangleq a.(s.\mathbf{0} \mid P)$.
2. $W \xrightarrow{\tau} \cdots$, sendo $W \triangleq a.W \mid W[\overline{a}/a]$.

Exercício I.16

Uma *máquina de azar* é um dispositivo que por vezes se encontra em salas de jogos e que, externamente, disponibiliza três acções: m (entrada de uma moeda), $\overline{win}(x)$ (recepção de x moedas) e $\overline{loss}$ (perda da moeda). Considere a seguinte especificação de uma máquina deste tipo ([Sti95]):

$$IO \triangleq m.\overline{bank}.(lost.\overline{loss}.IO + rel(x).\overline{win}(x).IO)$$

$$B(n) \triangleq bank.\overline{max}(n+1).left(x).B(x)$$

$$Dc \triangleq max(z).(\overline{lost}.\overline{left}(z).Dc + \sum_{1 \leq x \leq z} \overline{rel}(x).\overline{left}(z-x).Dc)$$

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

1. Explique o comportamento do processo $M(n)$ e esboce o respectivo diagrama de sincronização.
2. Mostre que um jogador pode efectivamente ganhar uma detrminada quantia de moedas.
3. Mostre que um jogador pode perder o jogo (*i.e.*, não receber nada)
4. Poderá concluir desta especificação qual das duas situações anteriores é mais provável verificar-se? Porquê?

Exercício I.17

Considere (mais uma!) especificação de um *buffer*, onde m e s designam, respectivamente, uma mensagem e uma sequência de mensagens. Assuma o significado usual para pas funções len, :, $head$ e $tail$ sobre sequências.

$$B(s) \triangleq in(m).B(m:s) + (\text{ if } len(s) > 0 \text{ then } \overline{out}(head(s)).B(tail(s)))$$

1. Explique se o *buffer* tem ou não capacidade limitada e a que disciplina de ordenação obedece (*i.e.*, LIFO ou FIFO).
2. Altere a especificação apresentada com base nos seguintes requisitos informais:

   Devem ser considerados dois sinais adicionais f (de "flush") e t (de "stop"). Após receber um f, o buffer *começa a despejar todas as mensagens que tinha armazenadas e, enquanto o faz, não pode aceitar novas entradas. Após receber um t, o* buffer *pode continuar a aceitar mais mensagens, mas não lhe é permitido que as transmita. Dois segundos depois regressa ao estado normal de operação, aceitando e transmitindo mensagens.*

   Especifique este novo *buffer*. Tenha o cuidado de prever a situação em que o *buffer* recebe um t enquanto está a despejar-se (em resposta a um f prévio). De que forma vai tratar o requisito *dois segundos depois ...* ?

Exercício I.18

Considere a seguinte especificação de um sistema de comunicação com possibilidade de perda de mensagens.

$$T \triangleq ok.send(x).\overline{s}(x).T$$

$$R \triangleq r(x).\overline{receive}(x).R$$

$$M \triangleq \overline{ok}.s(x).(\overline{r}(x).M + \tau.M)$$

$$S \triangleq (T \mid M \mid R)\backslash_{\{ok,s,r\}}$$

1. Mostre que, de facto, se podem perder mensagens.
2. Converta a especificação para a linguagem base $\mathbb{P}$, assumindo que as mensagens consideradas são caracteres.

Exercício I.19

Considere a seguinte especificação de uma fila de valores booleanos Q, com uma acção de sinalização $\overline{no}$ que indica fila vazia.

$$QB \triangleq (Q_1 \mid Q_2)\backslash m$$

$$Q_1 \triangleq in(x).\overline{m}(x).Q_1$$
$$Q_2 \triangleq m(x).\overline{out}(x).Q_2 + \overline{no}.Q_2$$

1. Esboce o respectivo diagrama de sincronização.
2. Calcule o grafo de transições correspondente.
3. Traduza a especificação para a linguagem base $\mathbb{P}$.
4. Defina um processo LG que tenta ler dois valores booleanos de QB e fornecer a sua conjunção ou disjunção, conforme pedido. Caso QB retorne $\overline{no}$ o valor comunicado por LG ao seu ambiente deve ser a constante $\perp$.
5. Componha em paralelo os processos QB e LG de forma a obter o comportamento esperado. Trace o diagrama de sincronização correspondente.
6. Determine a espécie sintática do processo composto.

Exercício I.20

Na sua expressão mais simples, uma Rede de Táxis é constituida por uma Central, que recebe solicitações de clientes potenciais para serviço imediato ou a agendar, e por um conjunto de viaturas que servem efectivamente os clientes por indicação da Central ou por sinalização directa destes. Do ponto de vista de um observador externo, a Rede de Táxis interactua com o seu ambiente atendendo telefonemas dos seus clientes e servindo-os, *i.e.*, iniciando e terminando corridas. Note-se, porém, que a arquitectura (informática) do sistema de gestão da rede segue de perto a sua configuração física, com um módulo activo na Central e pequenas unidades de comunicação nas viaturas. Detalhemos um pouco os requisitos operacionais para estes dois tipos de processos.

Viatura Uma viatura é caracterizada por um conjunto de eventos que modelam o início (sai, *i.e.*, sai da garagem) e fim (reg, *i.e.*, regressa) da sua actividade, assim como as diversas etapas compreendidas no serviço a clientes. Uma vez na rua, a viatura pode recolher um passageiro (rec) por sinalização deste ou em cumprimento de uma alocação anterior feita pela Central (alo). É possível que ao chegar ao ponto de recolha indicado pela Central o passageiro já não a aguarde, o que implicará a anulação do serviço (anu). Obviamente, uma vez iniciado um serviço, este deve em algum instante terminar, o que corresponderá a um evento que assinale o fim da corrida (fcr). Imediatamente após o início ou fim de uma corrida, a viatura deverá sinalizar a Central desses factos (snr e snf, respectivamente). Por outro lado, uma viatura ocupada poderá detectar a sinalização de um potencial cliente, que deverá comunicar à Central (det). Em qualquer momento a viatura terá capacidade para reportar à Central a sua localização corrente (tic).

Central O papel da Central na rede é duplo. Por um lado recebe, por via telefónica, solicitações de clientes que requerem um serviço imediato ou, simplesmente, pretendem reservar uma viatura para uma ocasião posterior. Por outro, mantém-se em ligação directa com todas as viaturas da rede acompanhando o desenrolar das actividades de cada uma. Em termos dinâmicos, o processo reconhece dois tipos de eventos: os que concernem à comunicação com as viaturas e os relativos ao atendimento dos clientes. Note-se que a alocação de uma viatura a um serviço é despoletada internamente sempre que foi atingida a hora marcada.

Especifique o processo que descreve a Rede de Táxis, respeitando os requisitos indicados, que, no entanto, poderá rever e sofisticar. Indique a razão das opções que vier a tomar.

Exercício I.21

Suponha que um colega seu justificou a equivalência entre os comportamentos exibidos pelos processos $I \triangleq (\text{ if } b \text{ then } P) \mid Q$ e $J \triangleq \text{ if } b \text{ then } (P \mid Q)$ usando o seguinte argumento: *Quando se faz a tradução de ambos para a linguagem base, o construtor condicional desaparece. Portanto, apenas permanecem as traduções de* P *e* Q.

Está de acordo? Em caso afirmativo tente fornecer uma prova formal, caso contrário exiba um contra-exemplo. (SUGESTÃO: use o facto, que provaremos mais tarde, de o processo $\mathbf{0}$ ser o elemento neutro da composição paralela.)

Capítulo 3

Cálculo de Processos

3.1 Motivação

As Questões

Neste capítulo vamos procurar responder à questão de saber em que circunstâncias dois processos, especificados na linguagem $\mathbb{P}$, representam o *mesmo* comportamento. Cada resposta possível conduzir-nos-á a uma definição de *equivalência* entre processos. E, certamente, dispôr de uma noção de equivalência representa um salto fundamental no nosso arsenal de ferramentas de concepção e análise de sistemas concorrentes.

De facto, ao especificar um dado processo, já, provavelmente, nos interrogamos sobre o que é essencial e acessório nessa descrição. Por exemplo, será relevante a ordem das parcelas num contexto aditivo? Intuitivamente diremos que não; mas em que é que baseamos essa resposta? E o que se passará com a composição paralela? Será justificável igualar $E \mid \mathbf{0}$ com E? E com $\tau.E$?

Analogamente, quando manipulamos expressões aritméticas sabemos que podemos reordenar as parcelas de uma soma sem que isso afecte o resultado. Mas sabemos mais: sabemos a razão que nos permite proceder dessa forma — *o resultado final da adição é o mesmo em qualquer caso.* No cálculo de processos ainda não temos um conceito semelhante. Poderíamos arriscar dizer que as *transições envolvidas são as mesmas*, o que, sem dúvida, bastaria para justificar a comutatividade da escolha. Mas já não seria suficiente para justificar $a.\tau.E = a.E$.

Uma definição matemática de equivalência entre processos irá permitir-nos abordar estas questões. Questões que se tornam muito relevantes na prática. Suponhamos, por exemplo, que necessitamos substituir uma componente de um sistema, porque esta se esgotou ou se tornou demasiado cara. Será seguro substitui-la por uma componente diferente? Sem dúvida, desde que o comportamento do sistema permaneça essencialmente o mesmo. Uma noção precisa de equivalência permitirá provar que a substituição é ou não segura.

Mas que noção de equivalência será esta? Sabemos em que condições dois autómatos finitos são equivalentes: sempre que reconhecem a mesma linguagem (regular). Mas será essa uma noção de equivalência aceitável entre processos? De facto, se pensarmos em processos como entidades dinâmicas que são observadas (*i.e.*, interagem) enquanto se executam, faz sentido tomar todos os estados por que passam como estados terminais, uma vez que não podemos interromper a observação em algum momento particular. Deveremos, então, identificar os processos que realizam as mesmas sequências de acções? Ou bastará requerer a igualdade dos seus prefixos finitos? Ou do seu contéudo observável? Mas será que os processos $a.(b.\mathbf{0}+c.\mathbf{0})$ e $a.b.\mathbf{0}+a.c.\mathbf{0}$, que tanto diferem na *quantidade de não determinismo*, são mesmo equivalentes? Ou a equivalência deve tomar em consideração não apenas as sequências de interacções mas também o seu padrão de ocorrência no tempo?

Por outro lado, teremos de reflectir sobre as propriedades que uma adequada noção de equivalência entre processos deve satisfazer. Dará origem a uma teoria equacional suficientemente expressiva? Será decidível? Será modular (*i.e.*, congruente)?

O Jogo

Um tipo de equivalência entre processos bastante fina (*i.e.*, que faz muitas distinções) pode ser estabelecida através de um jogo em que, dados dois processos E e F, dois jogadores discutem se as respectivas estruturas de transição são mutuamente correspondentes. Assim, o jogador I(nocêncio) tenta evidenciar essa correspondência, enquanto o jogador R(ezingão) procura mostrar o contrário.

Uma partida deste jogo é iniciada por R escolhendo uma transição $E \xrightarrow{a} E'$ (ou $F \xrightarrow{a} F'$). Atento, I tenta encontrar uma transição com a mesma etiqueta no outro processo. Caso o consiga, R volta a jogar, I tenta de novo acompanha-lo, e assim sucessivamente.

R ganha a partida se I fôr incapaz de encontrar uma transição com etiqueta igual à da escolhida por R. Por seu lado, I é vencedor sempre que, tendo respondido a todas as jogadas de R, se chegou a uma situação em que os processos em causa já apareceram antes no decorrer dessa partida, ou em que R não pode jogar (porque os processos em causa já não admitem mais transições). Ganha, igualmente, se a partida puder continuar indefinidamente.

Um jogador vence o jogo se fôr capaz de vencer todas as partidas iniciadas com E e F. Nesse caso diz-se que o jogador em questão tem uma *estratégia vencedora*. Por exemplo, em

$$B \triangleq in.B \mid \overline{out}.B$$
$$B' \triangleq inB' + \overline{out}.B'$$

suponhamos que R escolheu a transição $B \xrightarrow{in} B$. I responde com $B' \xrightarrow{in} B'$. R escolhe, agora, $B' \xrightarrow{\overline{out}} B'$, respondendo I com $B \xrightarrow{\overline{out}} B$, ... Note-se que a partida nunca termina uma vez que I possui uma estratégia vencedora que consiste em copiar sempre a última jogada de R.

Porque I possui uma estratégia vencedora dizemos que os processos B e B' são G-equivalentes. Notemos que uma estratégia vencedora para I significa, no caso geral, que

i) se R escolhe $E \xrightarrow{a} E'$ então I pode escolher $F \xrightarrow{a} F'$ e os processos E' e F' são ainda G-equivalentes.

ii) se R escolhe $F \xrightarrow{a} F'$ então I pode escolher $E \xrightarrow{a} E'$ e os processos E' e F' são ainda G-equivalentes.

Esta propriedade é conhecida por *bissimilariedade* e, como veremos, qualquer relação que a verifique é dita uma emphbissimulação em $\mathbb{P}$.

3.2 Bissimulação e Equivalência Estrita

Definições

Recordemos que uma relação é um conjunto de pares; no nosso caso um conjunto de pares de processos, ou, se preferirmos, um subconjunto do produto cartesiano $\mathbb{P} \times \mathbb{P}$. Existe certamente uma enorme variedade de relações definíveis sobre $\mathbb{P}$, a maioria das quais sem qualquer interesse. Por exemplo a relação "ter o mesmo número de conectivos que" ou a relação de identidade sintática. A discussão feita na secção anterior, porém, conduz-nos a um tipo muito especial de relações que vamos designar por *bissimulações* e que assumem uma enorme importância no cálculo de processos. Comecemos por introduzir a sua definição:

Definição 1 *Uma relação binária S em $\mathbb{P}$ é uma* bissimulação (estrita) *sse, sempre que $(E, F) \in S$ e $a \in Act$,*

i) *se $E \xrightarrow{a} E'$ então $F \xrightarrow{a} F'$ e $(E', F') \in S$*

ii) *se $F \xrightarrow{a} F'$ então $E \xrightarrow{a} E'$ e $(E', F') \in S$*

Notemos que esta definição qualifica uma propriedade sobre relações em $\mathbb{P}$: em verdade existem diversas relações que a satisfazem. Em particular, o leitor poderá verificar que

Lema 1 *Seja $\{S_i \subseteq \mathbb{P}\times\mathbb{P} |\ i \in I\}$ uma família de bissimulações estritas. Então, a identidade $\mathbf{1}_{\mathbb{P}}$, a relação vazia $\emptyset$, a inversa S_i^{-1} (para $i \in I$), a relação composta S_iS_j (para $i, j \in I$) e a união $\bigcup_{i\in I} S_i$ são, ainda, bissimulações estritas.*

Vejamos mais alguns exemplos de bissimulações. Claramente a relação $\{(\mathbf{0}, \mathbf{0})\}$ satisfaz a definição dada; a prova é trivial na medida em que $\mathbf{0}$ não admite quaisquer transições. Podemos também verificar sem grande esforço que $\{(a.\mathbf{0} + a.\mathbf{0}, a.\mathbf{0}), (\mathbf{0}, \mathbf{0})\}$ é uma bissimulação. Note-se que esta relação não é simétrica nem reflexiva, o que mostra que nem todas as bissimulações são relações de equivalência. Por outro lado, para uma relação ser bissimulação não é suficiente conter apenas pares de processos idênticos. Por exemplo, $\{(a.\mathbf{0}, a.\mathbf{0})\}$ falha a definição porque o processo $a.\mathbf{0}$ admite uma transição por a para $\mathbf{0}$, mas o par $(\mathbf{0}, \mathbf{0})$ não pertence à relação.

Qual será, pois, a característica fundamental de uma bissimulação? Como o leitor possivelmente já concluiu, trata-se de uma relação *fechada para a relação de transição*, no sentido em que as derivadas de processos relacionados devem estar igualmente relacionadas entre si. Neste facto reside a razão pela qual a equivalência entre processos se pode basear neste tipo de relações. Se existir uma bissimulação que contenha o par formado pelos processos E e F, podemos concluir que estes se podem simular mutuamente ao longo de todas as suas derivadas. A bissimulação em causa constitui, assim, uma *prova* de que os dois processos são indistinguíveis. Formalmente,

Definição 2 *Dois processos E e F são* estritamente equivalentes *(ou estritamente bissimilares), escrevendo-se $E \sim F$, sse existir uma bissimulação (estrita) S em $\mathbb{P}$ tal que $(E, F) \in S$.*

A definição dada é equivalente a dizer que

$$\sim\ =\ \bigcup\{S \subseteq \mathbb{P} \times \mathbb{P} |\ S \quad \text{é uma bissimulação (estrita)}\}$$

Recorrendo ao lema 1 podemos, agora, concluir que

Lema 2 *A relação $\sim$ constitui uma relação de equivalência e é a maior bissimulação definível em $\mathbb{P}$.*

Prova: Para $\sim$ ser uma relação de equivalência deverá ser reflexiva, simétrica e transitiva. A reflexividade decorre imediatamente do lema 1 (porquê?). Para provar a simetria, tomemos $E \sim F$, o que significa que existe uma bissimulação S tal que $(E, F) \in S$. Mas, então, $(F, E) \in S^{-1}$, que é ainda uma bissimulaçao de acordo com o lema referido. Logo $F \sim E$. Suponhamos, agora, que $E \sim F$ e $F \sim G$. Então existem duas bissimulações estritas S_1 e S_2 tais que $(E, F) \in S_1$ e $(F, G) \in S_2$. Logo, (E, G) pertence à composição $S_1 S_2$ que, pelo mesmo lema, é ainda uma bissimulação. Consequentemente $E \sim G$, o que prova a transitividade. Finalmente, o mesmo lema estabelece que a união de bissimulações forma uma bissimulação. A união de todas as bissimulações sobre $\mathbb{P}$, que define $\sim$, constitui, evidentemente, a maior de todas.

❏

Provas de Equivalência

A definição 2 fornece uma importante técnica de prova para demonstrar que dois processos são estritamente equivalentes: basta exibir uma bissimulação (estrita) que os contenha. Por exemplo, a equivalência $S \sim M$, para

$$\begin{aligned} T &\triangleq i.\overline{k}.T \\ R &\triangleq k.j.R \\ S &\triangleq (T \mid R)\backslash_{\{k\}} \end{aligned}$$

e

$$\begin{aligned} M &\triangleq i.\tau.N \\ N &\triangleq j.i.\tau.N + i.j.\tau.N \end{aligned}$$

decorre do facto $(S, M) \in R$, onde R é a seguinte bissimulação

$$R = \{(S, M), ((\overline{k}.T \mid R)\backslash_{\{k\}}, \tau.N), ((T \mid j.R)\backslash_{\{k\}}, N), ((\overline{k}.T \mid j.R)\backslash_{\{k\}}, j.\tau.N)\}$$

O leitor poderá facilmente verificar que, pelo contrário, não é possivel construir uma bissimulação contendo o par formado pelos processos $a.(b.\mathbf{0} + c.\mathbf{0})$ e $a.b.\mathbf{0} + a.c.\mathbf{0}$.

Vamos, agora, considerar um outro exemplo, recordando a especificação de um semáforo feita no capítulo anterior. Um semáforo é basicamente um dispositivo de controlo do acesso a um recurso critíco de acordo com a expressão

$$Sem \triangleq get.put.Sem$$

Suponhamos, agora, que, num problema, necessitamos de um semáforo n-ário, *i.e.*, um semáforo que admite qualquer sequência de *get* e *put* desde que a diferença entre o número dos primeiros e o número dos segundos não exceda n. Este tipo de semáforos é utilizado para controlar o acesso a recursos que admitem um número limitado de acessos simultâneos.

Uma especificação directa deste processo será, por exemplo,

$$\begin{aligned} Sem_n(0) &\triangleq get.Sem_n(1) \\ Sem_n(i) &\triangleq get.Sem_n(i+1) + put.Sem_n(i-1) \\ &\quad (\text{para } 0 < i < n) \\ Sem_n(n) &\triangleq put.Sem_n(n-1) \end{aligned}$$

Alternativamente, poderemos tentar realizar Sem_n pela composição paralela de n processos Sem, *i.e.*,

$$Sem^n \triangleq Sem \mid Sem \mid ... \mid Sem$$

Intuitivamente, cada *get* sobre Sem^n corresponde a um *get* de uma das suas componentes, pelo que se torna possível realizar tantas destas acções quanto o número de componentes agregadas em paralelo. Uma vez esgotado o número de semáforos livres o sistema não dará mais permissões de acesso até que pelo menos um deles seja libertado.

Não é difícil mostrar que $Sem_n(0) {\sim} Sem^n$. Por exemplo, para $n = 2$, chegamos rapidamente à seguinte bissimulação:

$$\begin{aligned} \{ &(Sem_2(0), Sem \mid Sem), (Sem_2(1), Sem \mid put.Sem), \\ &(Sem_2(1), put.Sem \mid Sem)(Sem_2(2), put.Sem \mid put.Sem)\} \end{aligned}$$

Notemos que o segundo e o terceiro pares de S são quase iguais: os lados esquerdos são-no efectivamente e os direitos são estritamente equivalentes (pois, como mostraremos adiante, a composição paralela é comutativa relativamente a $\sim$). No entanto, para S ser de facto uma bissimulação não nos é possível prescindir de qualquer destes pares: se o fizessemos S deixaria de ser fechada para a relação de transição.

Este tipo de redundância, tantas vezes presente nas bissimulações, pode ser explorado por uma técnica expedita para reduzir a quantidade de trabalho envolvida na demonstração de uma equivalência. Suponhamos, para ilustrar a ideia, que retirarmos de S o par $(Sem_2(1), Sem \mid put.Sem)$. A relação resultante, embora não satisfaça a definição 1, constitui uma *bissimulação a menos de* $\sim$, de acordo com

Definição 3 *Uma relação binária S em $\mathbb{P}$ é uma bissimulação (estrita) a menos de $\sim$ sse, sempre que $(E, F) \in S$ e $a \in Act$,*

i) *se $E \xrightarrow{a} E'$ então $F \xrightarrow{a} F'$ e $(E', F') \in {\sim}S{\sim}$*

ii) *se $F \xrightarrow{a} F'$ então $E \xrightarrow{a} E'$ e $(E', F') \in {\sim}S{\sim}$*

onde ${\sim}S{\sim}$ representa a composição das relações $\sim$, S e $\sim$, por esta ordem. I.é, $(E, F) \in {\sim}S{\sim}$ sse existirem processos E' e F' tais que $E \sim E'$, $(E', F') \in S$ e $F' \sim F$. Não se requer, pois, que S seja fechada para as derivadas, mas apenas que a cada derivada de um dos processos corresponda uma derivada do outro que conduza a um processo estritamente equivalente ao primeiro.

O interesse da definição 3 reside na simplificação que acaba por introduzir nas provas. Voltando ao exemplo dos semáforos, necessitaríamos apenas de um conjunto de 3, e não 4, pares de processos. O leitor poderá verificar que, para um n arbitrário, a bissimulação que justifica $Sem_n(0) \sim Sem^n$ contém 2^n pares enquanto uma bissimulação a menos de $\sim$ necessita apenas de $n+1$ pares, o que representa, para n elevado, uma economia razoável. É claro que para podermos usar esta noção no establecimento de $\sim$ teremos, primeiro, de demonstrar que

Lema 3 *Se S é uma bissimulação (estrita) a menos de $\sim$, então $S \subseteq \sim$.*

Dispomos, assim, de uma poderosa técnica de prova: para establecer $E \sim F$, para F e E arbitrarios, é suficiente exibir uma bissimulação a menos de $\sim$ que contenha o par (E, F). A demonstração do lema 3 baseia-se no seguinte resultado:

Lema 4 *Se S é uma bissimulação (estrita) a menos de $\sim$, então $\sim S \sim$ constitui uma bissimulação (estrita).*

A prova é feita diagramaticamente em [Mil89] (página 93). Para o leitor que tenha dificuldade com esse tipo de linguagem fica aqui a prova detalhada pela construção usual.

Prova: Seja S uma bissimulação (estrita) a menos de $\sim$. Queremos provar que $\sim S \sim$ satisfaz as cláusulas i) e ii) de definição 1. Vamos, pois, assumir que $E \sim S \sim F$, para algum par de processos E e F. Suponhamos que $E \xrightarrow{a} E'$. Então, porque $E \sim S \sim F$, existem processos E_1 e F_1 tais que $E \sim E_1$, $E_1 S F_1$ e $F_1 \sim F$. De $E \xrightarrow{a} E'$ e $E \sim E_1$ concluimos que existe um E_1' tal que $E_1 \xrightarrow{a} E_1'$ e $E' \sim E_1'$. Agora, $E_1 \xrightarrow{a} E_1'$ e $E_1 S F_1$, juntamente com o facto de S ser uma bissimulação a menos de $\sim$, permite-nos concluir que existe um F_1' que verifica $F_1 \xrightarrow{a} F_1'$ e $E_1' \sim S \sim F_1'$. Finalmente, se $F_1 \xrightarrow{a} F_1'$ e $F_1 \sim F$, então $F \xrightarrow{a} F'$ e $F' \sim F_1'$. Sumariando temos que, para algum E_1' e F_1' se verifica

$$F \xrightarrow{a} F' \qquad \text{e} \qquad E' \sim E_1' S F_1' \sim F'$$

A cláusula i) da definição de bissimulação segue daqui, dada a transitividade de $\sim$. A prova da cláusula ii) usa o argumento simétrico.

❏

Podemos, agora, verificar o lema 3 notando que, sendo, como acabamos de mostrar, $\sim S \sim$ uma bissimulação estrita, então $\sim S \sim \subseteq \sim$. Sabemos também que a relação de identidade em $\mathbb{P}$ está contida em $\sim$. Logo $S \subseteq \sim S \sim$ e, transitivamente, $S \subseteq \sim$, como queríamos.

Torna-se porém necessário algum cuidado na utilização de bissimulação a menos de $\sim$ como técnica de demonstração. De facto, ao usar $\sim$ não podemos igualar os processos cuja equivalência estamos a tentar demonstrar, o que resultaria numa prova circular. Teremos, pois, de nos limitar a processos previamente provados equivalentes, tendo o cuidado de não caír em argumentos circulares.

3.3 Teoria da Equivalência Estrita

Propriedades Básicas

Vamos, agora, debruçarmo-nos sobre um conjunto de leis satisfeitas pela relação $\sim$. Juntamente com os resultados das próximas secções, estas propriedades formam o núcleo do *cálculo* de processos que anunciamos no início desta disciplina e que R. Milner designou por CCS (*i.e.*, *calculus of communicating systems*).

Ficamos, assim, equipados não apenas com uma metodologia para descrever problemas em níveis de abstracção distintos (e progressivamente mais detalhados), mas também com um cálculo equacional para discutir a sua equivalência (por exemplo, quando queremos mostrar a correcção da implementação de um processo P relativamente a uma especificação E). O cálculo permite-nos, nomeadamente, substituir, na prova de $E \sim P$, a construção explícita de uma bissimulação contendo E e P, por um argumento equacional.

Lema 5 *Sejam $E, F, G \in \mathbb{P}$. Então, $(\mathbb{P}, +, \mathbf{0})$ e $(\mathbb{P}, \mid, \mathbf{0})$ são monoides abelianos, sendo $+$ uma operação idempotente. Detalhadamente,*

$$
\begin{aligned}
(E+F)+G &\sim E+(F+G)\\
E+F &\sim F+E\\
E+\mathbf{0} &\sim E\\
E+E &\sim E\\
(E \mid F) \mid G &\sim E \mid (F \mid G)\\
E \mid F &\sim F \mid E\\
E \mid \mathbf{0} &\sim E
\end{aligned}
$$

Prova: As provas destas igualdades seguem, todas elas, um padrão similar (a que o leitor se deverá habituar realizando-as como exercício). Começa-se por conjecturar uma relação que contenha os processos cuja equivalência se quer demonstrar. De seguida verifica-se se a relação escolhida é, de facto, uma bissimulação por conferência das condições da definição 1 para todos os pares envolvidos. O único cuidado a ter é com a conjectura inicial. Ilustrativamente tentemos demonstrar a idempotência da escolha. Vamos começar por tomar a relação $S = \{(E+E, E) | \ E \in \mathbb{P}\}$. Ao tentar verificar as condições de (1) concluimos que, apesar de uma transição $E+E \xrightarrow{a} E$ ser correspondida por $E \xrightarrow{a} E$, o par (E, E) não pertence a S. Necessitamos, pois, de acrescentar a S todos os pares deste tipo, *i.e.*, definimos S como $\{(E+E, E) | \ E \in \mathbb{P}\} \cup Id_{\mathbb{P}}$ que facilmente se verifica ser uma bissimulação. Note-se que aqui, como na maioria dos casos que encontraremos, a bissimulação que testemunha a equivalência procurada é uma relação infinita.

❏

Lema 6 *Sejam $E, F \in \mathbb{P}$, $K \subseteq L$ e f e g renomeações em L. Então,*

$$
\begin{array}{rll}
E\backslash_K \sim E & & \textit{se } \mathbb{L}(E) \cap (K \cup \overline{K}) = \emptyset\\
(E\backslash_K)\backslash_{K'} \sim E\backslash_{K \cup K'} & & \\
(E[f])\backslash_K \sim (E\backslash_{f^{-1}(K)})[f] & & \\
(E \mid F)\backslash_K \sim E\backslash_K \mid F\backslash_K & & \textit{se } \mathbb{L}(E) \cap \overline{\mathbb{L}(F)} \cap (K \cup \overline{K}) = \emptyset\\
E[\mathbf{1}_L] \sim E & & \\
E[f] \sim E[g] & & \textit{se } f \restriction \mathbb{L}(E) = g \restriction \mathbb{L}(E)\\
E[f][g] \sim E[g \circ f] & & \\
(E \mid F)[f] \sim E[f] \mid F[f] & & \textit{se } f \restriction (K \cup \overline{K}),\ \textit{onde } K = \mathbb{L}(E \mid F),\ \textit{fôr injectiva}
\end{array}
$$

Prova: A demonstração destas leis é feita, de novo, por construção de uma bissimulação (estrita) contendo os respectivos lados direito e esquerdo. Assim, por exemplo, para mostrar que

$$E\backslash_K \ \sim \ E \qquad \text{se } \mathbb{L}(E) \cap (K \cup \overline{K}) = \emptyset$$

construimos, para um $K \subseteq Act$ arbitrário, a relação

$$S \ = \ \{(E\backslash_K, E) | \ E \in \mathbb{P} \wedge \mathbb{L}(E) \cap (K \cup \overline{K}) = \emptyset\}$$

Para cada par $(E\backslash_K, E) \in S$, se E transita por a para E', então $a, \overline{a} \notin K$, por definição de S. Logo as transições de qualquer elemento do par são correspondidas por transições idênticas do outro. Por outro lado, $\mathbb{L}(E') \subseteq \mathbb{L}(E)$, porque E' é uma derivação de E. Logo $\mathbb{L}(E') \cap (K \cup \overline{K}) = \emptyset$ o que permite concluir que $(E'\backslash_K, E') \in S$. Donde S é uma bissimulação (estrita), o que justifica a lei.

❏

Um último comentário sobre as provas deste dois lemas torna-se necessário. Em alguns casos, nomeadamente nas leis que envolvem *apenas* operadores *estáticos*, a verificação pode resumir-se a concluir a igualdade entre os diagramas de sincronização dos respectivos lados direito e esquerdo. De facto, se não se verifica qualquer alteração nestes diagramas, que exibem as capacidades de interacção dos processos, podemos concluir a sua equivalência uma vez que estes operadores não são consumidos pelas transições. O argumento não se aplica, porém quando estão envolvidos operadores dinâmicos. Por exemplo $a.\mathbf{0}$ e $a.a.\mathbf{0}$ têm o mesmo diagrama de sincronização mas não são, obviamente, equivalentes.

Modularidade

Mostramos já que $\sim$ é uma relação de equivalência (lema 2). Mas será que a substituição de um processo E por um processo F tal que $E \sim F$, num qualquer contexto, produz ainda resultados equivalentes? Notemos que se tal não fôr válido será quase impossível utilizar a noção de equivalência na prática: apenas se poderiam aplicar as leis anteriores ao nível mais externo das expressões e não às suas sub-expressões. A resposta, felizmente, é afirmativa: $\sim$ é compatível com os operadores em $\mathbb{P}$, ou seja, forma uma *congruência* em $\mathbb{P}$. De facto,

Lema 7 *Seja $E \sim F$. Então, para $P \in \mathbb{P}$, $K \subseteq L$ e f uma renomeação em L,*

$$a.E \sim a.F$$
$$E + P \sim F + P$$
$$E \mid P \sim F \mid P$$
$$E\backslash_K \sim F\backslash_K$$
$$E[f] \sim F[f]$$

A prova segue o padrão das anteriores e é sugerida como exercício. Necessitamos ainda de dar mais um passo para estabelecer completamente o resultado que nos interessa. De facto, uma vez que admitimos a definição *recursiva* de processos, quer através do uso de constantes, quer pelo operador $\underline{fix}$ $(X = E(X))$, torna-se necessário mostrar que este tipo de definições preserva a equivalência estrita. Comecemos, pois, por estender $\sim$ a processos com variáveis. Suponhamos que E e F contém, no máximo, $\tilde{X}$ variáveis livres. Então, $E \sim F$ sse, para todo o vector de processos $\tilde{P}$, $E[\tilde{P}/\tilde{X}] \sim F[\tilde{P}/\tilde{X}]$. Neste contexto, tem-se

Lema 8

i) *Se $\tilde{P} \triangleq \tilde{E}$ então $\tilde{P} \sim \tilde{E}$, onde $\tilde{E}$ é uma família de expressões em $\mathbb{P}$ e $\tilde{P}$ uma família de constantes.*

ii) *Seja $\tilde{E} \sim \tilde{F}$, onde $\tilde{E}$ e $\tilde{F}$ são famílias de expressões recursivas em $\mathbb{P}$ sobre um vector $\tilde{X}$ de variáveis. Definam-se $\tilde{A} \triangleq \tilde{E}[\tilde{A}/\tilde{X}]$ e $\tilde{B} \triangleq \tilde{F}[\tilde{B}/\tilde{X}]$. Então, $\tilde{A} \sim \tilde{B}$,* i.e., *$\sim$ é preservada pela definição recursiva.*

Prova: A primeira parte é trivial. Pela regra semântica (D) da definição de constantes sabemos que, para cada i, E_i e P_i têm exactamente as mesmas derivadas, pelo que o resultado segue imdiatamente. A prova da segunda parte, porém, é mais complexa, usando indução sobre os diagramas de inferência e o conceito de bissimulação a menos de $\sim$ discutido atrás. O leitor é convidado a consulta-la em [Mil89], páginas 99 e seguintes, mas não deve desanimar se a não compreender à primeira — este tipo de demonstração não é essencial para o que se segue.

❑

O Teorema da Expansão

A teoria da equivalência estrita que começamos a desenvolver inclui, ainda, um resultado fundamental sobre a relação entre processos definidos em termos da composição paralela e processos definidos por escolha não determinista. Tal resultado é conhecido em CCS como o *teorema da expansão*, precisamente porque indica que a composição concorrente de processos pode ser expandida (*i.e.*, "desenrolada") numa escolha cujas parcelas são prefixadas pelas acções iniciais do processo composto.

Antes de enunciarmos a lei convirá recordar o exemplo atrás discutido onde concluimos, porventura com alguma surpresa, que $S \sim M$. Vejamos, agora, o resultado geral, cuja prova o leitor encontrará em [Mil89], páginas 96 e 97.

Lema 9 *Seja* $E \triangleq (E_1[f_1] \mid ... \mid E_n[f_n])\backslash_K$, *com* $n \geq 1$. *Então,*

$$\begin{aligned} E \sim & \sum\{f_i(a).\,(E_1[f_1] \mid ... \mid E'_i[f_i] \mid ... \mid E_n[f_n])\backslash_K \mid E_i \xrightarrow{a} E'_i \ \wedge\ f_i(a) \notin K \cup \overline{K}\} \\ & + \\ & \sum\{\tau.\,(E_1[f_1] \mid ... \mid E'_i[f_i] \mid ... \mid E'_j[f_j] \mid ... \mid E_n[f_n])\backslash_K \mid E_i \xrightarrow{a} E'_i \ \wedge\ E_j \xrightarrow{b} E'_j \ \wedge\ f_i(a) = \overline{f_j(b)}\} \end{aligned}$$

Fazendo, na expressão anterior, $f_1 = f_2 = ... = f_n = \mathbf{1}_{\mathbb{P}}$, chegamos a um caso particular do teorema com muito interesse. Assim,

$$\begin{aligned} E \sim & \sum\{a.\,(E_1 \mid ... \mid E'_i \mid ... \mid E_n)\backslash_K \mid E_i \xrightarrow{a} E'_i \ \wedge\ a \notin K \cup \overline{K}\} \\ & + \\ & \sum\{\tau.\,(E_1 \mid ... \mid E'_i \mid ... \mid E'_j \mid ... \mid E_n)\backslash_K \mid E_i \xrightarrow{a} E'_i \ \wedge\ E_j \xrightarrow{\overline{a}} E'_j\} \end{aligned}$$

Note-se, porém, que a formulação deste lema é notacionalmente pesada. Uma forma alternativa de pensar nele consiste em considerar o seguinte resultado (de resto, mais geral e, certamente, mais fácil de memorizar): *todo o processo é equivalente à soma das suas derivadas imediatas*, *i.e.*,

$$E \ \sim\ \sum\{a.E' \mid E \xrightarrow{a} E'\}$$

Outros casos particulares surgem no seguinte corolário do teorema da expansão.

Lema 10 *Para $E, F \in \mathbb{P}$, $a \in Act$, $K \subseteq L$ e f uma renomeação em $\mathbb{P}$, verifica-se*

$$\begin{aligned}
(E+F)[f] &\sim E[f] + F[f] \\
(E+F)\backslash_K &\sim E\backslash_K + F\backslash_K \\
(a.E)[f] &\sim f(a).E[f] \\
(a.E)\backslash_K &\sim \begin{cases} \mathbf{0} & se\ a \in (K \cup \overline{K}) \\ a.(E\backslash_K) & caso\ contrário \end{cases}
\end{aligned}$$

Prova: Os resultados seguem por aplicação directa do lema 9, fazendo $n = 1$ em todos os casos, $K = \emptyset$ no primeiro e terceiro e $f = \mathbf{1}_{Act}$ nos dois restantes.

❏

Vejamos, por fim, como o teorema se aplica ao exemplo anteriormente discutido, permitindo-nos deduzir por raciocínio equacional, $S \sim M$. Temos, pois,

$$\begin{aligned}
S &\sim (T \mid R)\backslash_{\{k\}} \\
&\sim i.(\overline{k}.T \mid R)\backslash_{\{k\}} \\
&\sim i.\tau.(T \mid j.R)\backslash_{\{k\}} \\
&\sim i.\tau.(i.\ (\overline{k}.T \mid j.R)\backslash_{\{k\}} + j.(T \mid R)\backslash_{\{k\}}) \\
&\sim i.\tau.(i.j.\ (\overline{k}.T \mid R)\backslash_{\{k\}} + j.i.(\overline{k}.T \mid R)\backslash_{\{k\}}) \\
&\sim i.\tau.(i.j.\tau.\ (T \mid j.R)\backslash_{\{k\}} + j.i.\tau.(T \mid j.R)\backslash_{\{k\}})
\end{aligned}$$

Designando por N' a expressão $(T \mid j.R)\backslash_{\{k\}}$ que, como vimos, expande em $N' {\sim} i.j.\tau.(T \mid j.R)\backslash_{\{k\}}$ $+ j.i.\tau.(T \mid j.R)\backslash_{\{k\}}$, podemos concluir as equivalências $N' \sim N$ e $S \sim i.\tau.N \sim M$ como queríamos[1].

O teorema da expansão, sustentando que a concorrência pode ser "explicada" em termos de não determinismo, tem um significado muito profundo no cálculo de processos que temos vindo a introduzir. De facto, neste cálculo a concorrência é reduzida à *intercalação não determinista* das sequências de acções em que o processo considerado se pode comprometer, tornando-se a sua caracterização semântica essencialmente uma teoria do não determinismo. Esta abordagem à semântica dos processos, que é comum a Ccs e outros cálculos (como Csp [Hoa85] ou Acp [BK85]), é justamente conhecida por *paradigma da intercalação* e é motivada pela hipótese de que a relação de causalidade entre acontecimentos não é observável. Ao sucesso deste paradigma não será alheio o facto de os sistemas de multiplexação de tarefas e utilizadores sobre uma mesma unidade de processamento ter sido uma das primeiras realizações da programação concorrente.

3.4 Equivalência Observacional

Nesta secção vamos definir uma noção de equivalência insensível às acções não observáveis, *i.e.*, que permite considerar equivalentes processos que diferem na quantidade de comportamento não observável que realizam. Veremos que este requisito tem de ser formulado cuidadosamente. Para isso comecemos por definir o que se entende por transições observáveis de um processo.

[1] Estas últimas substituições, parecendo intuitivas, são de facto justificadas por um resultado (o lema 18) sobre a existência de soluções únicas para equações recursivas que introduziremos mais adiante.

Transições Observáveis

A relação de transição básica que temos usado para caracterizar a semântica operacional dos processos é a família de relações $\longrightarrow \subseteq \mathbb{P} \times Act \times \mathbb{P}$. Similarmente, podemos definir

$$\stackrel{a}{\Longrightarrow} \subseteq \mathbb{P} \times \mathbb{P}$$

como uma família de relações ditas de *transição observável*, em que as etiquetas são elementos de $L \cup \{\epsilon\}$. Uma $\stackrel{\epsilon}{\Longrightarrow}$-transição corresponde a zero ou mais transições elementares não observáveis (*i.e.*, etiquetadas por τ). As seguintes regras de inferência definem a relação que nos interessa:

$$\frac{}{E \stackrel{\epsilon}{\Longrightarrow} E}\,(O_1)$$

$$\frac{E \stackrel{\tau}{\longrightarrow} E' \quad E' \stackrel{\epsilon}{\Longrightarrow} F}{E \stackrel{\epsilon}{\Longrightarrow} F}\,(O_2)$$

$$\frac{E \stackrel{\epsilon}{\Longrightarrow} E' \quad E' \stackrel{a}{\longrightarrow} F' \quad F' \stackrel{\epsilon}{\Longrightarrow} F}{E \stackrel{a}{\Longrightarrow} F}\,(O_3) \quad \text{para } a \in L$$

As duas relações de transição são facilmente estendidas a sequências finitas de acções. Assim, temos $\stackrel{\omega}{\longrightarrow} \subseteq \mathbb{P} \times \mathbb{P}$, com $\omega \in Act^*$, dada por,

$$\frac{}{E \stackrel{<>}{\longrightarrow} E}\,(S_1)$$

$$\frac{E \stackrel{a}{\longrightarrow} E' \quad E' \stackrel{\omega}{\longrightarrow} F}{E \stackrel{<a,\omega>}{\longrightarrow} F}\,(S_2)$$

Similarmente, se define a relação $\stackrel{\omega}{\Longrightarrow} \subseteq \mathbb{P} \times \mathbb{P}$, com $\omega \in L^*$, dada por

$$\frac{E \stackrel{\omega'}{\longrightarrow} F}{E \stackrel{\omega}{\Longrightarrow} F}\,(S_3)$$

se ω fôr a parte observável da sequência $\omega' \in Act^*$. Com base na relação $\Longrightarrow$ vamos reconstituir as definições de bissimulação e equivalência dadas atrás.

NOTA: Em [Mil89] a noção de transição observável é captada de modo ligeiramente diferente. Dada uma sequência $\omega \in Act^*$, começa-se por definir uma nova sequência $\hat{\omega} \in L^*$ obtida de ω por remoção de todas as ocorrências de τ. Seguidamente define-se, para $\omega = a_1 a_2 ... a_n \in Act^*$, a relação

$$\stackrel{\omega}{\Longrightarrow} \quad \text{como} \quad (\stackrel{\tau}{\longrightarrow})^* \stackrel{a_1}{\longrightarrow} (\stackrel{\tau}{\longrightarrow})^* ... \stackrel{a_n}{\longrightarrow} (\stackrel{\tau}{\longrightarrow})^*$$

Note-se que, enquanto $\stackrel{\omega}{\longrightarrow}$ considera exactamente todas as transições por τ em ω, $\stackrel{\omega}{\Longrightarrow}$ especifica pelo menos as transições por τ presentes em ω. Por seu lado, se tomarmos $\stackrel{\hat{\omega}}{\Longrightarrow}$ estamos a ignorar completamente esse tipo de transições. A relação que acima introduzimos pode ser definida a partir destas: $\stackrel{a}{\Longrightarrow}$, para $a \in L$, equivale a $\stackrel{\hat{a}}{\Longrightarrow}$, enquanto $\stackrel{\epsilon}{\Longrightarrow}$ é $\stackrel{<>}{\Longrightarrow}$.

Um Exemplo

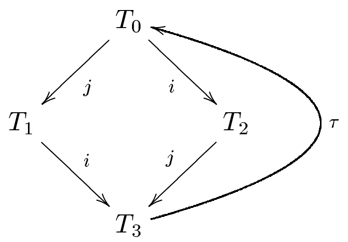
A importância de uma noção de equivalência em $\mathbb{P}$ baseada em transições observáveis pode ser ilustrada com o exemplo seguinte. Consideremos os processos

$$\begin{aligned} T_0 &\triangleq j.T_1 + i.T_2 \\ T_1 &\triangleq i.T_3 \\ T_2 &\triangleq j.T_3 \\ T_3 &\triangleq \tau.T_0 \end{aligned}$$

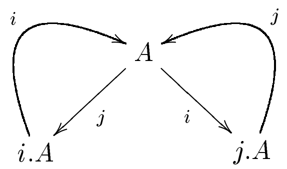
e

$$A \triangleq i.j.A + j.i.A$$

A análise dos seus grafos de transição



e



respectivamente, permite concluir de imediato que que $T_0 \not\approx A$. De facto, a transição $T_3 \xrightarrow{\tau} T_0$ não tem correspondente no processo A. No entanto, a transição que os distingue é etiquetada por uma acção não observável. Podemos, pois, tentar formular uma equivalência entre processos que seja "cega" para as transições não observáveis. Tal equivalência irá igualar, por exemplo, transições como $E \xrightarrow{a} E'$ com sequências de transições como $E \xrightarrow{\tau}\xrightarrow{\tau} \ldots \xrightarrow{a} E'$ ou $E \xrightarrow{a} \ldots \xrightarrow{\tau}\xrightarrow{\tau} E'$ ou, ainda, $E \xrightarrow{\tau}\xrightarrow{\tau} \ldots \xrightarrow{a} \ldots \xrightarrow{\tau}\xrightarrow{\tau} E'$. Todas estas sequências de transições são, como sabemos, abstraídas em $E \stackrel{a}{\Longrightarrow} E'$. Assim, o que temos de fazer é apenas refazer as definições de bissimulação e equivalência estritas em termos da relação $\Longrightarrow$.

Definições

Definição 4 *Uma relação binária S em $\mathbb{P}$ é uma* bissimulação *(não estrita, ou fraca, ou observacional) sse, sempre que $(E, F) \in S$ e $a \in L \cup \{\epsilon\}$,*

i) *se $E \stackrel{a}{\Longrightarrow} E'$ então $F \stackrel{a}{\Longrightarrow} F'$ e $(E', F') \in S$*

ii) *se* $F \stackrel{a}{\Longrightarrow} F'$ *então* $E \stackrel{a}{\Longrightarrow} E'$ *e* $(E', F') \in S$

Facilmente se pode verificar que, tal como acontecia no caso estrito, $\mathbf{1}_{\mathbb{P}}$, $\emptyset$, a inversa de uma bissimulação, a composição e a união de bissimulações, são, ainda, bissimulações de acordo com a definição acima. Podemos, agora, definir, à semelhança do que fizemos para $\sim$,

Definição 5 *Dois processos E e F são* (observacionalmente) equivalentes *(ou* bissimilares*), escrevendo-se $E \approx F$, sse existir uma bissimulação (observacional) S em $\mathbb{P}$ tal que $(E, F) \in S$.*

À luz desta definição podemos concluir, no exemplo anerior, que $T_0 \approx A$. Basta, para isso, mostrar que a relação

$$S = \{(T_0, A), (T_1, i.j.A), (T_2, j.i.A), (T_3, A)\}$$

é uma bissimulação (não estrita).

Propriedades

As propriedades de $\approx$ são similares às que encontramos para $\sim$. Em particular, a definição de $\approx$ equivale a

$$\approx \; = \; \bigcup\{S \subseteq \mathbb{P} \times \mathbb{P} |\; S \text{ é uma bissimulação}\}$$

o que permite concluir que $\approx$ é a maior bissimulação (não estrita). Constitui, além disso, uma relação de equivalência. Discutiremos mais adiante a preservação de $\approx$ pelos operadores da linguagem.

Uma propriedade fundamental desta relação, que não é obviamente partilhada por $\sim$, é traduzida na lei

Lema 11 *Para qualquer processo $E \in \mathbb{P}$,*

$$E \;\approx\; \tau.E$$

Prova: Para toda a acção a tal que $E \xrightarrow{a} E'$ temos $\tau.E \xrightarrow{\tau} E \xrightarrow{a} E'$. I.é, $\tau.E \stackrel{a}{\Longrightarrow} E'$. Por outro lado, a única acção $\tau.E \xrightarrow{\tau} E$ é correspondida pela acção nula $E \stackrel{\epsilon}{\Longrightarrow} E$ em E. Claramente $\mathbf{1}_{\mathbb{P}} \cup \{(E, \tau.E)|\; E \in \mathbb{P}\}$ é uma bissimulação (não estrita).

❏

Notemos, também, que, como se esperaria,

Lema 12 *Para quaisquer E e $F \in \mathbb{P}$, se $E \sim F$ então $E \approx F$,* i.e., $\sim \subseteq \approx$.

Por fim, tal como fizemos para o caso estrito, também aqui podemos definir a noção de *bissimulação a menos de* $\approx$, o que, de novo, reduz substancialmente o tamanho das relações que é necessário construir para provar uma equivalência.

Definição 6 *Uma relação S em $\mathbb{P}$ constitui uma bissimulação a menos de $\approx$ se $(E,F) \in S$ implica, para todo $a \in L$,*

i) *Se $E \xrightarrow{a} E'$ então existe um F' tal que $F \overset{a}{\Longrightarrow} F'$ e $E' \sim S \approx F'$.*

ii) *Se $F \xrightarrow{a} F'$ então existe um E' tal que $E \overset{a}{\Longrightarrow} E'$ e $E' \approx S \sim F'$.*

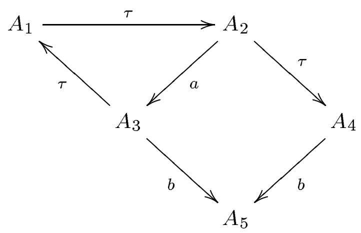
NOTA:

Construção de Bissimulações

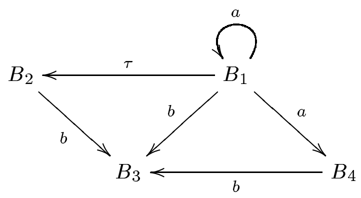
A construção de bissimulações constitui um procedimento necessário em muitas situações práticas. Justifica-se, por isso, um comentário um pouco extenso sobre um algoritmo de construção, aplicável a processos com um número finito de estados. Esta técnica baseia-se na análise dos grafos de transições e é, nomeadamente, usada no CWB (não sendo, contudo, mencionada em [Mil89]).

Consideremos, ilustrativamente os seguintes grafos de transições, que poderiam corresponder, por exemplo, aos processos

$$A_1 \triangleq \tau.(a.(b.\mathbf{0} + \tau.A_1) + \tau.b.\mathbf{0})$$
$$B_1 \triangleq a.B_1 + a.b.\mathbf{0} + \tau.b.\mathbf{0} + b.\mathbf{0}$$



e



O problema consiste em determinar quais os pares de estados bissimilares. A ideia desta técnica reside em procurar construir uma bissimulação iterativamente, a partir da relação universal sobre o conjunto de estados em causa, na qual todos os estados estão relacionados entre si. Desta forma temos a certeza de encontrar sempre a *maior* bissimulação contendo o par de processos em causa (caso exista, evidentemente!). Seguidamente, classificamos os estados de acordo com as suas derivadas imediatas para $\overset{a}{\Longrightarrow}$, para todo o $a \in L$. Estados equivalentes deverão ter as mesmas derivadas (observáveis).

Neste exemplo os estados A_1, A_2, A_3 e B_1 têm derivadas por $\overset{a}{\Longrightarrow}$ e por $\overset{b}{\Longrightarrow}$. Vamos agrupa-los num conjunto $C_1 = \{A_1, A_2, A_3, B_1\}$. Da mesma forma construimos os conjuntos $C_2 = \{A_4, B_2, B_4\}$ e $C_3 = \{A_5, B_3\}$, correspondendo aos estados que apenas têm derivadas por $\overset{b}{\Longrightarrow}$ (C_2) ou simplesmente não podem realizar qualquer transição observável (C_3).

Estes conjuntos definem uma relação binária nos estados: dois estados estão relacionados sse pertencerem ao mesmo conjunto. Vamos, agora, verificar se essa relação já será uma bissimulação. Como vimos, todo o estado em C_1 pode realizar $\overset{a}{\Longrightarrow}$ e evoluir para um estado também membro de C_1, facto que vamos registar escrevendo $C_1 \overset{a}{\Longrightarrow} C_1$. Da mesma forma concluimos que $C_1 \overset{a}{\Longrightarrow} C_2$, $C_1 \overset{\epsilon}{\Longrightarrow} C_2$, $C_1 \overset{b}{\Longrightarrow} C_3$ e $C_2 \overset{b}{\Longrightarrow} C_3$. Reparemos que cada uma destas asserções estabelece uma propriedade válida para todo o estado no conjunto à esquerda da seta; por outro lado, foram consideradas em cada caso todas as possíveis transições. Quer isto dizer que a relação *pertence ao mesmo conjunto* é, de facto, uma bissimulação, e, em particular $A_1 \approx B_1$. Note-se, por fim, que as transições por $\stackrel{\epsilon}{\Longrightarrow}$ dentro do mesmo conjunto (por exemplo, $A_3 \stackrel{\tau}{\longrightarrow} A_1$) podem ser sempre trivialmente simuladas, a partir de qualquer estado no conjunto, pela inacção.

No caso de a relação encontrada não ser ainda uma bissimulação, a construção continua por divisão de um dos conjuntos. Para ilustrar isto mesmo vamos considerar uma nova transição etiquetada por a entre A_1 e A_5. A classificação dos estados permanece a mesma, mas agora temos $A_1 \stackrel{a}{\Longrightarrow} A_5$ e $A_3 \stackrel{a}{\Longrightarrow} A_5$, enquanto $A_5 \in C_3$. Temos, então, $C_1 \stackrel{a}{\Longrightarrow} C_3$ apenas para alguns estados em C_1, nomeadamente A_1 e A_3. Logo C_1 não pode ser uma classe de equivalência de uma bissimulação. Vamos, pois, gerar um novo conjunto C_4 composto pelos estados de C_1 que podem alcançar estados de C_3 por $\stackrel{a}{\Longrightarrow}$. I.é, $C_4 = \{A_1, A_3\}$.

Temos, agora, que verificar se o resultado forma uma bissimulação. De facto, temos $C_4 \stackrel{a}{\Longrightarrow} C_3$ e $C_4 \stackrel{\epsilon}{\Longrightarrow} C_1$. Não basta, contudo, verificar o novo conjunto de estados, mas todos aqueles com transições para C_4 ou C_1, uma vez que estes conjuntos foram alterados. O conjunto C_1 tem de ser revisto porque $A_2 \stackrel{a}{\Longrightarrow} A_3$, que pertence a C_4, enquanto B_1 não pode atingir estados em C_4. Torna-se, então, necessário dividir de novo o conjunto C_1, originando $C_1 = \{B_1\}$ e $C_5 = \{A_2\}$. As transições a partir de C_4 são, agora, $C_4 \stackrel{a}{\Longrightarrow} C_3$ e $C_4 \stackrel{a}{\Longrightarrow} C_4$ (através de C_5), $C_4 \stackrel{a}{\Longrightarrow} C_5$, $C_4 \stackrel{a}{\Longrightarrow} C_2$, $C_4 \stackrel{\epsilon}{\Longrightarrow} C_5$, $C_4 \stackrel{\epsilon}{\Longrightarrow} C_2$ e $C_4 \stackrel{b}{\Longrightarrow} C_3$. Os conjuntos C_5 e C_1 têm apenas um estado, não constituindo, portanto, problema. Os conjuntos C_2 e C_3 não possuem transições conducentes a estados dos conjuntos que foram divididos; estando correctos antes da divisão assim continuam. Concluimos, pois, que a nova relação é uma bissimulação.

Prosseguindo desta forma acabamos sempre por obter uma bissimulação, pois, no limite, o processo de divisão termina em conjuntos de estados singulares e a relação identidade é trivialmente uma bissimulação. O ponto importante é que obtemos sempre a *maior* bissimulação, na medida em que todas as tentativas maiores são eliminadas pelo processo de divisão. Assim, temos a certeza que estados no mesmo conjunto são observacionalmente equivalentes e, dualmente, estados em conjuntos distintos não o podem ser, porque, efectivamente, a maior bissimulação relaciona todos os estados equivalentes. No segundo exemplo considerado, temos, em particular $A_1 \not\approx B_1$.

Este método de construção de bissimulações oferece-nos, ainda, um bónus importante. De facto, se tomarmos cada conjunto C_i como nodo de um grafo cujos arcos são transições a partir de cada elemento de cada C_i, obtemos um grafo de transições equivalente mas com um número *minímo* de estados. No primeiro exemplo (sem a transição adicional por a) obtemos:

$$\begin{tikzcd} C_1 \arrow[loop above, "a"] \arrow[r, "\tau{,}a"] \arrow[dr, "b"'] & C_2 \arrow[d, "b"] \\ & C_3 \end{tikzcd}$$

Este grafo é mínimo em número de estados mas não no que concerne ao número de transições. De facto, as duas transições $C_1 \stackrel{a}{\longrightarrow} C_2$ e $C_1 \stackrel{b}{\longrightarrow} C_3$ são redundantes, *i.e.*, simuláveis pelas restantes (como?). Se as removermos, chegamos a um grafo mínimo quer em termos de estados quer de transições:

$$\begin{tikzcd} C_1 \arrow[loop above, "a"] \arrow[r, "\tau"] & C_2 \arrow[r, "b"] & C_3 \end{tikzcd}$$

O processo correspondente a C_1 pode ser escrito como

$$M \;\triangleq\; a.M + \tau.b.\mathbf{0}$$

Como o processo A_1 pertence a este conjunto, obtivemos, assim, uma representação minimal (e observacionalmente equivalente) a A_1.

Como referimos, os algoritmos de verificação de equivalência e minimização utilizados no Cwb operam exactamente deste modo. A mesma técnica pode ser usada para a construção de bissimulações estritas (e determinação de $\sim$) se se considerarem transições da forma $\stackrel{a}{\longrightarrow}$ em vez de $\stackrel{a}{\Longrightarrow}$.

3.5 Congruência Observacional

O Problema

Tal como acima argumentamos, para ser útil no raciocínio sobre processos a equivalência $\approx$ deveria ser preservada pelos operadores do cálculo. Veremos nesta secção que tal preservação falha para um desses operadores, o que nos motivará a introduzir uma pequena, mas importante, modificação no conceito de equivalência observacional. Comecemos por notar que, tal como acontece para $\sim$, temos que

Lema 13 *Seja $E \approx F$. Então, para $P \in \mathbb{P}$, $K \subseteq L$ e f uma renomeação em L,*

$$\begin{aligned} a.E &\approx a.F \\ E \mid P &\approx F \mid P \\ E\backslash_K &\approx F\backslash_K \\ E[f] &\approx F[f] \end{aligned}$$

No entanto, nas condições do lema anterior, já não se verifica $E + P \approx F + P$. I. é, a substituição de um processo E por um processo F, tal que $E \approx F$, num contexto aditivo, não leva a resultados $\approx$-equivalentes. Dito de outro modo, $\approx$ *não* é uma congruência.

Um exemplo ajudará a perceber o que se passa. Seja $P \triangleq i.\mathbf{0}$ e $Q \triangleq \tau.i.\mathbf{0}$. $P \approx Q$, uma vez que podemos igualar Q a $\tau.P$ e, como vimos, $E \approx \tau.E$, para todo o processo E. No entanto, não se verifica

$$j.\mathbf{0} + P \approx j.\mathbf{0} + Q$$

De facto, os grafos de transição

$$\begin{tikzcd} & j.\mathbf{0} + P \arrow[dl, "j"'] \arrow[dr, "i"] & \\ \mathbf{0} & & \mathbf{0} \end{tikzcd}$$

e

$$\begin{tikzcd} & j.\mathbf{0} + Q \arrow[dl, "j"'] \arrow[dr, "\tau"] & \\ \mathbf{0} & & i.\mathbf{0} \arrow[d, "i"'] \\ & & \mathbf{0} \end{tikzcd}$$

dos processos compostos, indicam que o segundo pode evoluir invisivelmente para um estado onde apenas pode realizar i. I. é, a transição $j.\mathbf{0} + Q \stackrel{\epsilon}{\Longrightarrow} i.\mathbf{0}$ corresponde à transição $j.\mathbf{0} + P \stackrel{\epsilon}{\Longrightarrow} j.\mathbf{0} + P$, mas claramente $i.\mathbf{0}$ e $j.\mathbf{0} + P$ não são bissimilares.

O exemplo anterior ilustra bem a razão pela qual $\approx$ não é uma congruência: o problema reside no τ inicial, em Q, que, no contexto aditivo, restringe o "menu" de opções possíveis. Somos, assim, levados à seguinte definição,

Definição 7 *Dois processos E e F são observacionalmente congruentes (ou iguais), escrevendo-se $E = F$, sse*

i) $E \approx F$

ii) *se* $E \xrightarrow{\tau} E'$ *então* $F \xrightarrow{\tau} F'' \overset{\epsilon}{\Longrightarrow} F'$ *e* $E' \approx F'$

iii) *se* $F \xrightarrow{\tau} F'$ *então* $E \xrightarrow{\tau} E'' \overset{\epsilon}{\Longrightarrow} E'$ *e* $E' \approx F'$

A intuição sobre esta definição é de que toda a acção *inicial* não observável de E e F deve ser correspondida por, pelo menos, uma acção não observável do outro processo. É fácil mostrar que $=$ é uma congruência, inclusivamente para os contextos aditivos, e que, como seria de esperar, $E \neq \tau.E$. Tem-se, contudo, $\tau.E = \tau.\tau.E$ porque, após a transição inicial por τ, obtemos o par $(\mathbf{0}, \tau.\mathbf{0})$ de processos observacionalmente equivalentes. Repare-se que a definição de $=$ acima não é recursiva, nem é feita directamente em termos de bissimulações. De facto, as duas cláusulas da definição, com o requisito de que um τ deve ser correspondido por, pelo menos, outro τ, apenas são válidas ao primeiro nível das expressões; a partir daí usa-se a definição de equivalência observacional. Dito de outro modo, o requisito nas derivadas é formulado em termos de $\approx$ e não de $=$.

Propriedades

A propriedade fundamental da relação $=$ reside, por um lado, na sua insensibilidade para toda a acção invísivel (não inicial!), que herda da definição de $\approx$, e, por outro, na sua preservação pelos operadores em $\mathbb{P}$, em particular por $+$. De facto, um resultado muito interessante estabelece $=$ como uma restricção de $\approx$ aos pares de processos que a preservam em *todos* os contextos aditivos. Esta caracterização ilustra, igualmente, a proximidade entre as duas relações. A prova não é totalmente trivial e merece alguma atenção (parte dela é remetida para exercício em [Mil89]).

Lema 14 *Dados dois processos E e F cuja reunião das espécies é distinta do conjunto L de nomes de acções, $E = F$ sse, para todo o $G \in \mathbb{P}$,*

$$E + G \approx F + G$$

Prova: Comecemos por supôr que $E = F$. Basta-nos, pois, mostrar que

$$S \triangleq \{(E+G, F+G) |\ E, F, G \in \mathbb{P}\} \cup \approx$$

forma uma bissimulação. Temos de provar que S satisfaz as duas cláusulas da definição 4. Mostremos a primeira (a segunda segue o argumento simétrico). Suponhamos que $(U, V) \in S$ e que $U \xrightarrow{a} U'$. Se $U \approx V$ então $V \overset{a}{\Longrightarrow} V'$ e $U' \approx V'$ (porque $\approx$ é uma bissimulação não estrita), o que implica $(U', V') \in S$. Suponhamos, agora, e esta é a hipótese interessante, que U é da forma $E + G$, V é da forma $F + G$, com $E = F$. Há, neste caso, duas possibilidades de obter a transição $U \xrightarrow{a} U'$. A primeira consiste em $G \xrightarrow{a} G'$, mas, neste caso, tem-se igualmente $F + G \xrightarrow{a} G'$, *i.e.*, $V \xrightarrow{a} G'$ e, como $G' \approx G'$, tem-se $(G', G') \in S$, como queríamos. A segunda possibilidade consiste em $E \xrightarrow{a} E'$. Então, de $E = F$, segue-se que $F \overset{a}{\Longrightarrow} F'$, com $E' \approx F'$. Logo, $F + G \overset{a}{\Longrightarrow} F'$ e $(E', F') \in S$.

Mostremos, agora, a implicação reversa por contra-recíproco (*i.e.*, da negação da tese procuremos concluir a negação da hipótese). Suponhamos, pois, que $E \neq F$. Isto significa que existe pelo menos um a e um F' tais que $F \xrightarrow{a} F'$ mas, se $E \overset{a}{\Longrightarrow} E'$, então $E' \not\approx F'$. Vamos escolher $G \triangleq x.\mathbf{0}$, para x não pertencente nem à espécie de E nem de F. Notemos que é aqui que se torna necessária a hipótese do enunciado sobre as espécies de E e F. De facto, desconhece-se se o resultado é válido sem a hipótese. Note-se que, dada a admissibilidade em $\mathbb{P}$ de somas infinitas, os processos E e F poderiam, sem ela, usar todos os nomes de acções em L, mesmo para L infinito.

Prosseguindo, temos, pois, a transição $F + G \stackrel{a}{\Longrightarrow} F'$. Vamos, então, mostrar que, sempre que $E + G \stackrel{a}{\Longrightarrow} E'$, se tem $E' \not\approx F'$. Se $a = \tau$ e E' é da forma $E + G$, então o resultado segue porque E' pode realizar a acção x e F' não. Em qualquer outro caso, tem-se $E + G \stackrel{a}{\Longrightarrow} E'$, logo $E \stackrel{a}{\Longrightarrow} E'$, uma vez que a transição $G \stackrel{a}{\Longrightarrow} E'$ é impossível dado $a \neq x$. Então $E' \not\approx F'$.

❏

Com base neste lema é muito simples provar a preservação de $=$ em contextos aditivos (que $\approx$ falha), *i.e.*, $E + G = F + G$, desde que $E = F$.

Prova: A prova reduz-se, pelo lema anterior, a mostrar que, para todo o $G' \in \mathbb{P}$,

$$(E + G) + G' \;\approx\; (F + G) + G'$$

Sabemos que $E + (G + G') \approx F + (G + G')$, por aplicação do mesmo lema a $E = F$. Por outro lado, a escolha não determinística é associativa para $\approx$, na medida em que já o era para a equivalência estrita $\sim$ que está contida em $\approx$. Logo a equação acima verifica-se.

❏

Esta última prova remete-nos para a possibilidade de hierarquizar as várias noções de equivalência que introduzimos. Assim,

Lema 15 *Para quaisquer E e $F \in \mathbb{P}$, se $E \sim F$ então $E = F$. Similarmente, se $E = F$ então $E \approx F$,* i.e., $\sim \subseteq = \subseteq \approx$.

Note-se, porém, que da definição de $=$ decorre que, sendo E e F processos *estáveis*, *i.e.*, incapazes de realizar τ como primeira acção, $E \approx F$ implica $E = F$. Reparemos, ainda, que $\approx$ pode ser caracterizada em termos da igualdade, de acordo com o resultado seguinte,

Lema 16 *Para quaisquer E e $F \in \mathbb{P}$, dizemos que $E \approx F$ sse $E = F$ ou $E = \tau.F$ ou $\tau.E = F$.*

Em resumo, a relação $=$, que vamos tomar como a noção de *igualdade* em $\mathbb{P}$, é uma congruência (inclusivamente com respeito à definição recursiva, caso em que se prova um resultado idêntico, mas formulado em termos de $=$, ao lema 8). Sem surpresas, a teoria equacional subjacente incorpora, de resto, todas as leis que estudamos para a equivalência estrita, incluindo o teorema da expansão. Além disso, tem-se especificamente

Lema 17 *Para quaisquer $E, F \in \mathbb{P}$ e $a \in L$ verificam-se as seguintes igualdades que permitem controlar a ocorrência de τ,*

$$\begin{aligned} a.\tau.E &= a.E \\ E + \tau.E &= \tau.E \\ a.(E + \tau.F) &= a.(E + \tau.F) + a.F \end{aligned}$$

cuja prova decorre trivialmente da definição 7.

Notemos, por fim, que, apesar da motivação e esforço feito na procura de uma congruência observacional, a simples equivalência ($\approx$) não deixa de ser útil. E isto porque não só $\approx$ é aplicável em muitas situações, mas sobretudo porque tem associada, tal como a equivalência estrita ($\sim$), uma poderosa técnica de prova — a construção de uma bissimulação contendo o par de processos em análise — que é automatizável para uma grande classe de processos (em particular, para processos de estado finito). Além disso os lemas anteriores mostram que, ao raciocinar sobre a igualdade de processos, podemos utilizar $\approx$ e introduzir $=$ apenas no "último passo" (porquê?).

Solução de Equações

É agora altura de discutir um resultado a que já fizemos referência e que concerne à *existência de soluçoes únicas* para equações recursivas nas estruturas $(\mathbb{P}, \sim)$ e $(\mathbb{P}, =)$. Em particular, o lema seguinte estabelece que tais soluções existem para sistemas de n equações recursivas sobre n variáveis em $\mathbb{P}$, desde que a ocorrência das variáveis satisfaça determinadas condições. A prova do lema não é fundamental para este curso (mas pode ser consultada em [Mil89], páginas 158 a 160); mais importante é não confundir este resultado com o lema 8, nem com o seu correspondente para a congruência observacional. Um (este!) é sobre *equações* recursivas, o outro concerne às *definições* recursivas e cai, consequentemente, no grupo de propriedades necessárias para o estabelecimento de uma congruência. Temos, pois,

Lema 18 *Equações recursivas em $\mathbb{P}$ da forma $\tilde{X} = \tilde{E}(\tilde{X})$ ou $\tilde{X} \sim \tilde{E}(\tilde{X})$ têm soluções únicas em $\mathbb{P}$ (a menos de $=$ ou $\sim$, respectivamente). Em rigor,*

i) *Seja $\tilde{E} = \{E_i |\ i \in I\}$ uma família de expressões com, no máximo, I variáveis livres ($\{X_i |\ i \in I\}$) tais que toda a variável que ocorre livremente numa expressão E_i é fracamente guardada. Então, se $\tilde{P} \sim \tilde{E}[\tilde{P}/\tilde{X}]$ e $\tilde{Q} \sim \tilde{E}[\tilde{Q}/\tilde{X}]$, tem-se que $\tilde{P} \sim \tilde{Q}$.*

ii) *Seja $\tilde{E} = \{E_i |\ i \in I\}$ uma família de expressões com, no máximo, I variáveis livres ($\{X_i |\ i \in I\}$) tais que toda a variável que ocorre livremente numa expressão E_i é guardada e sequencial. Então, se $\tilde{P} = \tilde{E}[\tilde{P}/\tilde{X}]$ e $\tilde{Q} = \tilde{E}[\tilde{Q}/\tilde{X}]$, tem-se que $\tilde{P} = \tilde{Q}$.*

Precisemos as condições sobre as variáveis. Uma variável é *fracamente guardada* numa expressão E sse ocorre numa sub-expressão de E da forma $a.E'$, para $a \in Act$. X é dita *guardada* sse a, na condição anterior, representar uma acção observável (*i.e.*, $a \in L \cup \overline{L}$). Por exemplo, X é fracamente guardada em $\tau.X$ e $\tau.\mathbf{0} + a.X + b.a.X$, mas guardada apenas na segunda expressão. As condições em causa estabelecem, portanto, que o comportamento do processo é independente do processo que instancia a variável até à ocorrência de uma guarda.

No entanto, para equações recursivas em $=$ é ainda necessário impedir que a variável acabe por ficar guardada por um τ que resulte de uma sincronização. Tal seria o caso, por exemplo, de X em $X = (\overline{a}.X \mid a.\mathbf{0})\backslash_{\{a\}}$. Impõe-se, assim, que toda a sub-expressão estrita de E (*i.e.*, excluindo o próprio E) em que X ocorre seja da forma $a.E'$, para $a \in Act$, ou $\Sigma\tilde{E}$. Neste caso a ocorrência da variável diz-se *sequencial* na expressão E. Vejamos, de seguida, dois exemplos de aplicação deste resultado.

Exemplos

Relembremos o tratamento do problema da exclusão mútua através de um semáforo:

$$\begin{aligned}
Sem &\triangleq get.put.Sem \\
P_1 &\triangleq \overline{get}.c_1.\overline{put}.P_1 \\
P_2 &\triangleq \overline{get}.c_2.\overline{put}.P_2 \\
S &\triangleq (Sem \mid P_1 \mid P_2)\backslash_{\{get,put\}}
\end{aligned}$$

e consideremos, agora, um processo $S' \triangleq \tau.c_1.S' + \tau.c_2.S'$ cuja equivalência com S queremos discutir.

Comecemos por reparar que S' é solução da equação $X = E(X)$, onde $E(X) = \tau.c_1.X + \tau.c_2.X$ é uma expressão sobre a variável X. De facto, pelo primeiro lema, $S' = E[S'/X]$. Tentemos verificar se S é igualmente uma solução da equação em causa. Assim, e começando por aplicar o teorema da expansão, obtemos:

$$\begin{aligned}
S =\ & \tau.\,(c_1.\overline{put}.P_1 \mid P_2 \mid put.Sem)\backslash_K + \tau.(P_1 \mid c_2.\overline{put}.P_2 \mid put.Sem)\backslash_K \\
=\ & \tau.c_1.\,(\overline{put}.P_1 \mid P_2 \mid put.Sem)\backslash_K + \tau.c_2.(P_1 \mid \overline{put}.P_2 \mid put.Sem)\backslash_K \\
=\ & \tau.c_1.\tau.\,(P_1 \mid P_2 \mid Sem)\backslash_K + \tau.c_2.\tau.(P_1 \mid P_2 \mid Sem)\backslash_K \\
=\ & \tau.c_1.\tau.S + \tau.c_2.\tau.S \\
=\ & \tau.c_1.S + \tau.c_2.S \\
=\ & E[S/X]
\end{aligned}$$

o que permite concluir $S = S'$, por aplicação do lema (18).

Vejamos, agora, um exemplo um pouco mais trabalhoso. Consideremos os seguintes processos:

$$\begin{aligned}
B &\triangleq in.B_1 \\
B_1 &\triangleq in.B_2 + \overline{out}.B \\
B_2 &\triangleq \overline{out}.B_1
\end{aligned}$$

e

$$\begin{aligned}
B' &\triangleq (C_1 \mid C_2)\backslash_m \\
C_1 &\triangleq in.\overline{m}.C_1 \\
C_2 &\triangleq m.\overline{out}.C_2
\end{aligned}$$

Claramente o processo B é solução do seguinte sistema de 3 equações a 3 incógnitas (X, Y e Z):

$$\begin{aligned}
X &= E(X,Y,Z) = in.Y \\
Y &= E_1(X,Y,Z) = in.Z + \overline{out}.X \\
Z &= E_3(X,Y,Z) = \overline{out}.Y
\end{aligned}$$

através da substituição $\sigma = [B/X, B_1/Y, B_2/Z]$. I.é, $\tilde{B} = \tilde{E}[\sigma]$, dados os vectores $\tilde{B} = (B, B_1, B_2)$ e $\tilde{E} = (E, E_1, E_2)$.

Para estabelecer a equivalência $B = B'$, vamos, agora, procurar uma substituição σ' tal que $\tilde{B'} = \tilde{E}[\sigma']$, para um $\tilde{B'} = (B', S_1, S_2)$. A questão que se coloca é saber como transformar a descrição do processo B' de forma a ser possível identificar as expressões para S_1 e S_2 em σ'. Para isso vamos recorrer, de novo, ao teorema da expansão. Assim,

$$\begin{aligned} B' &= (C_1 \mid C_2)\backslash_m \\ &= in.(\overline{m}.C_1 \mid C_2)\backslash_m \\ &= in.\tau.(C_1 \mid \overline{out}.C_2)\backslash_m \\ &= in.(C_1 \mid \overline{out}.C_2)\backslash_m \end{aligned}$$

Tentativamente, façamos $S_1 = (C_1 \mid \overline{out}.C_2)\backslash_m$ e prossigamos a expansão:

$$\begin{aligned} S_1 &= (C_1 \mid \overline{out}.C_2)\backslash_m \\ &= in.\,(\overline{m}.C_1 \mid \overline{out}.C_2)\backslash_m + \overline{out}.(C_1 \mid C_2)\backslash_m \\ &= in.\,(\overline{m}.C_1 \mid \overline{out}.C_2)\backslash_m + \overline{out}.B' \end{aligned}$$

Façamos, por fim, $S_2 = (\overline{m}.C_1 \mid \overline{out}.C_2)\backslash_m$. Então,

$$\begin{aligned} S_2 &= (\overline{m}.C_1 \mid \overline{out}.C_2)\backslash_m \\ &= \overline{out}.(\overline{m}.C_1 \mid C_2)\backslash_m \\ &= \overline{out}.\tau.(C_1 \mid \overline{out}.C_2)\backslash_m \\ &= \overline{out}.\tau.S_1 \\ &= \overline{out}.S_1 \end{aligned}$$

A substituição pretendida fica, assim, completa, o que nos permite concluir a equivalência em causa entre B e B'.

Note-se que este problema pode ser reduzido ao da determinação de soluções de um sistema de 2 equações a 2 incógnitas, a saber,

$$\begin{aligned} X &= E(X, Y) = in.Y \\ Y &= E'(X, Y) = in.\overline{out}.Y + \overline{out}.in.Y \end{aligned}$$

Claramente, por substituição, obtemos

$$\begin{aligned} B &= in.B_1 \\ B_1 &= in.\overline{out}.B_1 + \overline{out}.in.B_1 \end{aligned}$$

Mostramos já que $B' = ... = in.S_1$. Prosseguindo o cálculo desta sub-expressão, obtemos

$$\begin{aligned} S_1 &= (C_1 \mid \overline{out}.C_2)\backslash_m \\ &= in.\,(\overline{m}.C_1 \mid \overline{out}.C_2)\backslash_m + \overline{out}.B' \\ &= in.\overline{out}.\,(\overline{m}.C_1 \mid C_2)\backslash_m + \overline{out}.B' \\ &= in.\overline{out}.\tau.\,(C_1 \mid \overline{out}.C_2)\backslash_m + \overline{out}.B' \\ &= in.\overline{out}.\tau.S_1 + \overline{out}.B' \\ &= in.\overline{out}.S_1 + \overline{out}.B' \\ &= in.\overline{out}.S_1 + \overline{out}.in.S_1 \end{aligned}$$

Logo, $B' = E[B'/X, S_1/Y]$ e $S_1 = E'[B'/X, S_1/Y]$. É imediato concluir que $B = B'$.

3.6 Análise de Casos

Vamos concluir este capítulo analisando alguns exemplos de raciocínio equacional sobre processos. Os casos a abordar ilustram situações típicas na especificação e implementação de sistemas sobre plataformas computacionais envolvendo distribuição e concorrência.

Especificação *versus* Implementação

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

NOTA: O uso dos termos *especificação* e *implementação* neste contexto pode parecer um pouco forçado na medida em que o *critério de correcção* que se adopta é uma equivalência, tipicamente a congruência observacional $=$. O aumento de *redundância* na passagem da especificação para a implementação não tem pois a ver com o comportamento observável dos processos, que permanece inalterado, mas antes com o grau de paralelismo (que aumenta).

Uma ordem de redundância comportamental em $\mathbb{P}$ pode ser obtida considerando a existência, não de uma bissimulação, mas de uma *simulação* do processo que constitui a especificação no processo implementador. Existem diversos tipos de simulação (a definição mais óbvia consiste em escolher apenas uma das cláusulas na definição de bissimulação) entre sistemas de transição [WN95] que podem ser utilizadas na ordem procurada. Note-se que, no caso geral, uma relação de implementação (ou *reificação*) forma uma pre-ordem cujo fecho anti-simétrico coincide com a noção de isomorfismo no universo matemático em causa [Oli90].

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

$$(E_1[f_1] \mid E_2[f_2] \mid \ldots \mid E_n[f_n])\backslash_K$$

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

Particularmente interessante são os casos em que a estrutura concorrente não é fixa de uma vez por todas mas evolutiva. Conseguem-se, assim, exprimir algumas situações em que o sistema se reconfigura dinamicamente. O caso mais simples é o da terminação de um ou mais factores numa composição paralela (*e.g.* em $(P \mid a.....\mathbf{0} \mid Q)\backslash_K$), que diminui o número de factores na composição. Por outro lado, a invocação recursiva de um processo concorrente num dos seus próprios factores provoca uma "explosão" de paralelismo, eventualmente originando sistemas com conjuntos de estados não finitos. É o caso de $\underline{fix}\ (X = (\ldots + a.X \mid \overline{a}.X + \ldots)\backslash_a)$. Interessam-nos, sobretudo, versões controladas dessa "explosão" que nos forneçam implementações distribuídas dos sistemas que concebemos.

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

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

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

O Π-*calculus*[MPW92, PT97] é uma tentativa de acomodar esta noção numa extensão adequada de CCS. A ideia base é a inclusão de nomes de ligações entre processos como parâmetros das acções, abolindo-se a distinção entre os identificadores de portas (acções) e os dados. Obtem-se, deste modo, uma teoria de primeira ordem para exprimir propriedades de ordem superior.

Contadores

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

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

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

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

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

onde o operador $\frown$ se define, entre dois processos E e F tais que $\{up, dw, zr, \overline{d}, \overline{z}\} \subseteq \mathbb{L}(E)$ e $\{up, dw, zr\} \subseteq \mathbb{L}(F)$, como

$$
\begin{aligned}
E \frown F \;=\; & (E[\overline{u'}/u, \overline{z'}/\overline{z}, \overline{d'}/\overline{d}] \mid F[u'/up, z'/zr, d'/dw])\backslash_A \\
& \text{onde } \; A = \{u', z', d'\}
\end{aligned}
$$

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

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

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

$$
\begin{aligned}
Z \frown Z \;&=\; (Z[\overline{u'}/u, \overline{z'}/\overline{z}, \overline{d'}/\overline{d}] \mid Z[u'/up, z'/zr, d'/dw])\backslash_A \\
&=\; (Z[\overline{u'}/u, \overline{z'}/\overline{z}, \overline{d'}/\overline{d}])\backslash_A \mid (Z[u'/up, z'/zr, d'/dw])\backslash_A \\
&=\; Z \mid \mathbf{0} \\
&=\; Z
\end{aligned}
$$

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

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

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

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

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

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

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

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

❏

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

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

Temos portanto

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

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

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

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

Stacks & Afins

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

$$
\begin{aligned}
St(\epsilon) &\triangleq push(x).St(x) + empty.St(\epsilon) \\
St(a:s) &\triangleq push(b).St(b:a:s) + \overline{pop}(a).St(s)
\end{aligned}
$$

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

$$\begin{aligned} C(x) &\triangleq push(y).(C(y) \frown C(x)) + \overline{pop}(x).P \\ P &\triangleq o(x).C(x) + \overline{e}.Z \\ \\ Z &\triangleq push(y).(C(y) \frown Z) + empty.Z \end{aligned}$$

onde o operador $\frown$ se define agora como

$$\begin{aligned} E \frown F \;&=\; E[\overline{u'}/u, \overline{e'}/\overline{e}, o'/o] \mid F[u'/push, e'/empty, \overline{o'}/\overline{pop}]\backslash_A \\ &\text{onde} \;\; A = \{u', e', o'\} \end{aligned}$$

Uma stack com a sequência $a_1 a_2 a_3 ... a_n$ como valor corrente será representada pela conexão de n células C, cada uma armazenando um dos valores considerados, e um elemento terminal Z representando a stack vazia. Ou seja,

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

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

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

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

Sistemas "Indutivos"

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

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

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

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

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

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

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

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

As componentes base C e Z definem-se por

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

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

Exercícios

Exercício II.1

Indique quais das seguintes relações em $\mathbb{P}$ são bissimulações estritas.

$$S_1 = \{(\mathbf{0}, \mathbf{0})\}$$
$$S_2 = \emptyset$$
$$S_3 = \{(a.\mathbf{0}, a.\mathbf{0} + \overline{a}.\mathbf{0}), (\mathbf{0}, \mathbf{0})\}$$
$$S_4 = \{(a.\mathbf{0}, a.\mathbf{0})\}$$
$$S_5 = \{(a.\mathbf{0} \mid b.\mathbf{0}, a.b.\mathbf{0} + b.a.\mathbf{0}), (\mathbf{0} \mid b.\mathbf{0}, b.\mathbf{0}), (a.\mathbf{0} \mid \mathbf{0}, a.\mathbf{0}), (\mathbf{0} \mid \mathbf{0}, \mathbf{0})\}$$

Exercício II.2

Sejam $X \triangleq receive.send.X$ e $Y \triangleq send.receive.Y$. Mostre que $\{(send.X, Y), (X, receive.Y)\}$ é uma bissimulação estrita. Poderá concluir que $X \sim Y$?

Exercício II.3

Indique quais dos seguintes processos são estritamente equivalentes a $a.b.\mathbf{0}$ (realize as provas e forneça os contra-exemplos necessários para justificar as suas conclusões.

1. $a.(b.\mathbf{0} + b.\mathbf{0})$
2. $a.b.\mathbf{0} + a.b.\mathbf{0}$
3. $a.\tau.b.\mathbf{0}$
4. $a.b.\mathbf{0} + a.\mathbf{0}$
5. $a.(b.\mathbf{0} + b.\mathbf{0}) + a.b.\mathbf{0}$
6. $a.b.\mathbf{0} + \mathbf{0}$
7. $a.\mathbf{0} \mid \mathbf{0}$
8. $a.(b.\mathbf{0} \mid c.\mathbf{0})\backslash_{\{c\}}$
9. $a.b.(c.\mathbf{0} \mid d.\mathbf{0})\backslash_{\{c,d\}}$

Exercício II.4

Para os processos seguintes determine processos estritamente equivalentes com um número mínimo de estados:

1. $A \triangleq a.B + a.D$, com $B \triangleq b.C$, $C \triangleq b.B$ e $D \triangleq b.C$.
2. $A \triangleq a.B + a.D$, com $B \triangleq a.D + b.C$, $C \triangleq b.A$ e $D \triangleq a.B + b.C$.
3. $A \triangleq a.A_2 + a.A_6$, com $A_2 \triangleq a.A_3$, $A_3 \triangleq a.A_4 + a.A_5$, $A_4 \triangleq a.A_5$, $A_5 \triangleq a.A_6$ e $A_6 \triangleq a.A$.

Exercício II.5

Suponha que alguém adicionou à linguagem de processos $\mathbb{P}$ dois outros operadores de composição paralela definidos pelas regras seguintes:

$$\frac{E \xrightarrow{a} E'}{E \otimes F \xrightarrow{a} E' \otimes F}\,(O_1) \qquad \frac{F \xrightarrow{a} F'}{E \otimes F \xrightarrow{a} E \otimes F'}\,(O_2)$$

$$\frac{E \xrightarrow{a} E' \quad \wedge \quad \overline{a} \notin \mathbb{L}(F)}{E \parallel F \xrightarrow{a} E' \parallel F}\,(P_1) \qquad \frac{F \xrightarrow{a} F' \quad \wedge \quad \overline{a} \notin \mathbb{L}(E)}{E \parallel F \xrightarrow{a} E \parallel F'}\,(P_2)$$

$$\frac{E \xrightarrow{a} E' \quad F \xrightarrow{\overline{a}} F'}{E \parallel F \xrightarrow{\tau} E' \parallel F'}\,(P_3)$$

1. Explique informalmente o propósito de $\otimes$ e $\parallel$, distinguindo-os da composição paralela que estudou.
2. A partir destas regras explique de que forma os diagramas de sincronização de $E \otimes F$ e $E \parallel F$ podem ser construidos a partir dos diagramas de E e de F. Compare esse processo com o que se passa com a construção do diagrama de sincronização de $E \mid F$.
3. Mostre ou refute a associatividade de $\parallel$ relativamente a $\sim$.

Exercício II.6

Determine um processo P tal que $P \mid (a.b.\mathbf{0}) \sim a.b.a.\mathbf{0} + a.a.b.\mathbf{0}$, ou mostre que um tal P não pode existir.

Exercício II.7

Considere o processo $A \triangleq a.(A \mid b.\mathbf{0})$.

1. Calcule o conjunto de derivadas de A.
2. Prove que $A \mid A \sim A$.

Exercício II.8

Mostre que, sendo S uma bissimulação a menos de $\sim$ e $(E, F) \in S$, se pode concluir que $E \sim F$.

Exercício II.9

Suponha que alguém forneceu a prova seguinte da comutatividade da composição paralela relativamente a $\sim$:

Prova: Vamos provar que $E \mid F \sim F \mid E$ por indução no tamanho de E e F. O caso de base, em que ambos os processos são $\mathbf{0}$, é trivial. Para o passo indutivo, suponhamos que $E \mid F \xrightarrow{a} G$. Temos de considerar três possibilidades para esta acção: pode tratar-se de uma acção de E, ou de F, ou de uma comunicação entre ambos. Na primeira hipótese G é da forma $E' \mid F$, com $E \xrightarrow{a} E'$. Mas, neste caso, tem-se igualmente $F \mid E \xrightarrow{a} F \mid E'$ e, por hipótese de indução, $E' \mid F \sim F \mid E'$. A segunda possibilidade segue um argumento simétrico. Na terceira hipótese tem-se $a = \tau$ e $E \mid F \xrightarrow{\tau} E' \mid F'$ pela regra de inferência $(C3)$. Igualmente se tem, contudo, $F \mid E \xrightarrow{\tau} F' \mid E'$ e $E' \mid F' \sim F' \mid E'$ é válido por hipótese de indução.

❏

Explique porque é que esta demonstração está errada e forneça a prova correcta.

Exercício II.10

Para os seguintes pares de processos indique, justificando, os que podem ser relacionados por $\approx$. E por $=$?

1. $a.\tau.b.\mathbf{0}$ e $a.b.\mathbf{0}$
2. $a.(b.\mathbf{0} + \tau.c.\mathbf{0})$ e $a.(b.\mathbf{0} + c.\mathbf{0})$
3. $a.(b.\mathbf{0} + \tau.c.\mathbf{0})$ e $a.(b.\mathbf{0} + c.\mathbf{0}) + a.c.\mathbf{0}$
4. $a.\mathbf{0} + b.\mathbf{0} + \tau.b.\mathbf{0}$ e $a.\mathbf{0} + \tau.b.\mathbf{0}$
5. $a.\mathbf{0} + b.\mathbf{0} + \tau.b.\mathbf{0}$ e $a.\mathbf{0} + b.\mathbf{0}$
6. $a.(b.\mathbf{0} + (\tau.(c.\mathbf{0} + \tau.d.\mathbf{0})))$ e $a.(b.\mathbf{0} + (\tau.(c.\mathbf{0} + \tau.d.\mathbf{0}))) + a.(c.\mathbf{0} + \tau.d.\mathbf{0})$
7. $a.(b.\mathbf{0} + (\tau.(c.\mathbf{0} + \tau.d.\mathbf{0})))$ e $a.(b.\mathbf{0} + c.\mathbf{0} + d.\mathbf{0}) + a.(c.\mathbf{0} + d.\mathbf{0}) + a.d.\mathbf{0}$
8. $\tau.(a.b.\mathbf{0} + a.c.\mathbf{0})$ e $\tau.a.b.\mathbf{0} + \tau.a.c.\mathbf{0}$
9. $\tau.(a.\tau.b.\mathbf{0} + a.b.\tau.\mathbf{0})$ e $a.b.\mathbf{0}$
10. $\tau.(\tau.a.\mathbf{0} + \tau.b.\mathbf{0})$ e $\tau.a.\mathbf{0} + \tau.b.\mathbf{0}$
11. $A \triangleq a.\tau.A$ e $B \triangleq a.B$
12. $A \triangleq \tau.A + a.\mathbf{0}$ e $a.\mathbf{0}$
13. $A \triangleq \tau.A$ e $\mathbf{0}$

Exercício II.11

Suponha que os processos R e T têm, entre outras, as transições seguintes: $R \xrightarrow{\tau} T$ e $T \xrightarrow{\tau} R$. Mostre que, nessa condições, se tem $R = T$.

Exercício II.12

Considere a definição seguinte de um *activador de n portas*:

$$
\begin{aligned}
AC(0) &\triangleq \overline{s}.\mathbf{0} \\
AC(n) &\triangleq a.AC(n-1)
\end{aligned}
$$

1. Explique o comportamento deste processo.
2. Defina um activador de 4 portas usando apenas cópias do processo $AC(2)$ e os operadores estáticos da linguagem.
3. Esboce o grafo de transições para esse processo e minimize-o colapsando os estados observacionalmente equivalentes.

4. A partir da resposta à questão anterior discuta se o processo que definiu pode ser considerado observacionalmente equivalente a $AC(4)$.

Exercício II.13

Mostre que $E \approx \tau.E$, exibindo uma bissimulação (observacional) contendo esse par de processos. Qual é a menor bissimulação (observacional) que contém todos os pares desse tipo ([Mil89], página 113) ?

Exercício II.14

Considere os factos seguintes acerca de uma relação binária S sobre $\mathbb{P}$. Para cada um deles, indique, justificando, se é ou não possível concluir que S constitui uma bissimulação observacional:

1. S é a relação identidade em $\mathbb{P}$.
2. S é um subconjunto da relação identidade em $\mathbb{P}$.
3. S é uma bissimulação estrita a menos de $\sim$.
4. S é uma bissimulação observacional a menos de $\approx$.
5. S é a relação vazia.
6. $S = \{(a.E, a.F)|\ E \approx F\}$.
7. $S = \{(a.E, a.F)|\ E \approx F\} \cup \approx$.

Exercício II.15

Mostre que

1. $E + \tau.(E+F) \ = \ \tau.(E+F)$
2. $a.(E + \tau.\tau.E) \ = \ a.E$
3. $\tau.(G + a.(E+\tau.F)) \ = \ \tau.(G + a.(E + \tau.F)) + a.F$

Exercício II.16

Considere a equação $X \ = \ a.\mathbf{0} + \tau.X$. Mostre que qualquer processo da forma $\tau.(\tau.P + a.\mathbf{0})$ é solução da equação. Considere, agora, o sistema de equações

$$\begin{aligned} X &= a.\mathbf{0} + \tau.Y \\ Y &= b.\mathbf{0} + \tau.X \end{aligned}$$

Como caracterizaria suas soluções ([Mil89], página 66) ?

Exercício II.17

Considere a equação $X \ = \ (a.X \mid \overline{a}.\mathbf{0})\backslash_{\{a\}}$, onde a variável X ocorre guardada, mas não sequencial, no lado direito. Mostre que o processo $\tau.P$ é solução desta equação, para qualquer P desde que $a, \overline{a} \notin \mathbb{L}(P)$ ([Mil89], página 67).

Exercício II.18

Para todo o processo E tal que $\mathbb{L}(E) = \emptyset$, prove ou refute as seguintes afirmações:

1. $E \mid Q \approx Q$.
2. $E \mid Q = Q$.
3. $E \mid Q = \tau.Q$.

Exercício II.19

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

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

onde

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

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

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

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

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

Exercício II.20

Seja $E \triangleq a(x).\overline{a}(x).E$, *i.e.*, E é um *buffer* de uma posição que utiliza a mesma porta para entrada e saída de valores. Assuma que esses valores são números inteiros.

1. Defina um processo sequencial (*i.e.*, sem recurso à composição paralela) F tal que $F \approx E \mid E$.
2. Prove essa equivalência construindo uma bissimulação fraca relacionando os dois processos.
3. Determine se a sua proposta para F é minimal, no sentido de não conter acções não observáveis desnecessárias.

Exercício II.21

Um *repetidor* é um processo definido como

$$R \triangleq a(x).R(x)$$
$$R(x) \triangleq \overline{z}(x).R(x) + a(y).R(y)$$

Assuma que o universo de valores para este processo se restringe aos booleanos.

1. Esboce o grafo de transições de R.
2. Seja $E \frown F \stackrel{\text{abv}}{=} (E[m/z] \mid F[m/a])\backslash_{\{m\}}$. Esboce o grafo de transições de $R \frown R$.
3. Indique quais os estados desse grafo que são observacionalmente equivalentes e desenhe o grafo de transições mínimo correspondente.

Exercício II.22

Considere os factos seguintes e, para cada um deles, indique, justificando, se é ou não possível concluir que $E = F$:

1. $E \approx F$ e E é um processo estável.
2. $E \approx F$ e nem E nem F são processos estáveis.
3. Existe um G tal que $E \mid G = F \mid G$.
4. $a.E = a.F$.
5. E e F satisfazem a mesma equação $X \sim E(X)$, onde as ocorrências de X em E são todas guardadas e sequenciais.

Exercício II.23

Considere um novo operador entre processos, dito *duplicador de acções*, e definido pela seguinte regra de inferência:

$$\frac{E \xrightarrow{a} E'}{\circlearrowleft (E) \xrightarrow{a} E}$$

Note que a derivada que surge na conclusão da regra é E e não E'. Por exemplo, $\circlearrowleft (a.\mathbf{0}) \xrightarrow{a} a.\mathbf{0}$. Prove ou refute que:

1. $E \sim F$ implica $\circlearrowleft (E) \sim \circlearrowleft (F)$.
2. $E \approx F$ implica $\circlearrowleft (E) \approx \circlearrowleft (F)$.
3. $\circlearrowleft (E + F) \sim \circlearrowleft (E) + \circlearrowleft (F)$.

Exercício II.24

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

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

Capítulo 4

Lógicas para Processos

4.1 Processos e Propriedades

Motivação

Até este ponto do curso foi nossa preocupação definir uma linguagem (*i.e.*, uma álgebra) para descrever processos comunicantes e discutir em que condições dois processos podem ser considerados equivalentes. Elaboramos, assim, uma teoria *equacional* que é *adequada* (*i.e.*, dois processos provados iguais na teoria — ou seja, por aplicação das suas leis — são semanticamente equivalentes) e *completa* para a classe dos processos com estados finitos (*i.e.*, dois processos semanticamente equivalentes podem ser provados iguais na teoria). No capítulo anterior ilustramos a aplicação deste tipo de raciocínio equacional à verificação de implementações distribuídas de processos sequenciais não determinísticos. Variantes à aproximação adoptada aqui podem ser obtidas por modificação dos conectivos base da linguagem (por exemplo, introduzindo primitivamente um operador de atraso ou permitindo sincronizações n-árias em $|$) e/ou pela definição de diferentes relações de equivalência sobre os termos da linguagem.

NOTA: Apesar de não prosseguirmos aqui nessa direcção, é importante salientar a enorme diversidade de *álgebras de processos* propostas a partir dos trabalhos pioneiros de R. Milner em CCS [Mil89] e C. A. R. Hoare em CSP [Hoa85]. A escolha entre diferentes conjuntos de conectivos ou interpretações semânticas, tem sobretudo a ver com a área de aplicação em causa e o nível de abstracção desejado. Por exemplo, uma pequena reformulação de CCS que permite a co-ocorrência de eventos revela-se adequada à especificação de sistemas síncronos [Mil83], *i.e.* sistemas cujas componentes não podem permanecer adormecidas a menos que realizem uma acção (τ) correspondente a esse estado. Os processos síncronos evoluem em *lock-step*, realizando *globalmente* um único evento em cada instante. A referência [Mil86] é uma excelente discussão sobre o que está em causa no desenho de uma álgebra de processos.

Referimos, numa nota anterior, o Π-*calculus* [MPW92], que permite exprimir estruturas de conexão dinamicamente modificáveis. Igualmente interessantes são as álgebras que consideram restricções temporais absolutas (*e.g.* "a ocorre 10ms após a ocorrência de b") com aplicação nos sistemas de tempo real (ver, por exemplo, [BB91]) e as que se baseiam em equivalências probabilísticas [LS91].

O problema que vamos abordar agora é, contudo, outro. No projecto de sistemas concorrentes é, por vezes, muito importante registar certas propriedades sobre o seu comportamento cuja satisfação deve ser garantida pelos processos que os implementam. Um conjunto de propriedades constitui uma especificação (eventualmente parcial) de um comportamento que, não se preocupando com todos os detalhes da sua descrição, regista os requisitos críticos que o cliente espera ver satisfeitos. Por exemplo, no projecto de um protocolo de comunicações o contrato entre o implementador e o cliente deve especificar que toda a mensagem transmitida é recebida, sem, contudo, descrever os mecanismos que garantem a sua verificação.

Suponhamos que pretendemos especificar a propriedade seguinte sobre uma viatura de uma rede de táxis (*cf.*, exercício I.20):

$\phi_0 =$ *A viatura pode recolher um passageiro ou ser alocada pela Central a um serviço pendente.*

Intuitivamente esta propriedade só será válida para uma viatura em serviço, *i.e.*, que previamente saiu da Central para as ruas. De facto, a noção de validade que nos interessa não é universal, mas *relativa* a um determinado estado da vida do processo que modela a viatura. A lógica que procuramos deverá, pois, ser capaz de exprimir esta *relatividade da verdade* em função dos estados alcançados pelo evoluir dos processos.

O universo que nos interessa é obviamente o conjunto $\mathbb{P}$ dos processos. Notemos, porém, que $\mathbb{P}$ é mais que um simples conjunto. De facto, está equipado com uma família de relações de transição que especificam de que forma um processo se transforma noutro pela ocorrência de uma determinada acção em Act. Os elementos de $\mathbb{P}$ constituem, como vimos atrás, os estados de um sistema de transição etiquetado por Act. Então certas propriedades poderão ser válidas num determinado estado e deixarem de o ser após a realização de uma transição. Por exemplo, a propriedade $\phi_1 =$ *O processo pode realizar uma transição por a* é verificada pelo processo $E \triangleq a.\mathbf{0} + b.c.\mathbf{0}$ mas já não pelo processo $\mathbf{0}$ que resulta da ocorrência de a. Este, por sua vez, verifica a propriedade $\phi_2 =$ *O processo está inactivo*, *i.e.*, nenhuma acção pode ser realizada.

Como poderemos exprimir a propriedade $\phi_3 =$ *O processo pode evoluir pela ocorrência de uma acção a para um processo inactivo*? Para isso necessitamos de introduzir na lógica um conectivo que estabeleça a validade de uma propriedade, não no estado representado pelo processo E, mas num estado alcançado por uma possível transição, etiquetada por a, a partir de E. Vamos representar esse conectivo por $\langle a \rangle$. A propriedade ϕ_3 escrever-se-á, então, como

$$\langle a \rangle \phi_2$$

Consideremos, agora, uma propriedade ligeiramente diferente: $\phi_4 =$ *O processo evolui por toda a ocorrência de uma acção a para um processo inactivo.* I.é, qualquer transição etiquetada por a impede o processo de prosseguir a sua actividade. Reparemos que E acima verifica ϕ_4, mas já $E' \triangleq a.\mathbf{0} + a.c.\mathbf{0}$ o não faz, uma vez que existe uma transição por a que conduz a $c.\mathbf{0}$ que, por sua vez, pode ainda realizar c. Há, em ϕ_4, uma quantificação universal sobre as transições por a: enquanto ϕ_3 se pode ler como *é possível uma transição por a que conduz a um estado que verifique ϕ_2*, ϕ_4 dever-se-á ler como *é necessário que uma transição por a conduza a um estado que verifique ϕ_2*. Ou, dito de outro modo, *toda* a transição por a conduz a um estado que verifica ϕ_2. Vamos exprimir isto recorrendo a um novo conectivo, dito de *necessidade*, e representado por $[a]$. A propriedade ϕ_4 escrever-se-á, então, como

$$[a]\phi_2$$

Note-se que $[a]$ e $\langle a \rangle$ são conectivos *duais*, no sentido em que o são os quantificadores universal e existencial na lógica de primeira ordem [1]. Atente-se, ainda, que nem todos os processos que verificam $[a]\phi_2$ verificam igualmente $\langle a\rangle\phi_2$. É, por exemplo, o caso do processo $F \triangleq d.\mathbf{0} + b.c.\mathbf{0}$ que falha $\langle a\rangle\phi_2$ por não exibir nenhuma transição por a conducente a $\mathbf{0}$, mas verifica $[a]\phi_2$ (exactamente pela mesma razão pela qual todos os habitantes da Lua são licenciados em Matemática).

Não há razão para nos restringirmos a acções individuais na parametrização deste dois novos conectivos. De facto, podemos parametriza-los por qualquer subconjunto não vazio de Act. Por exemplo a propriedade ϕ_5 = *O processo evolui por toda a ocorrência de a ou b para um processo inactivo* será expressa pela fórmula $[\{a, b\}]\phi_2$, que, por convenção, abreviamos para $[a, b]\phi_2$.

Pensemos, agora, de que forma podemos exprimir ϕ_2. Dizer que um processo é inactivo, significa que todas as transições a partir dele conduzem a estados "impossíveis", *i.e.*, que não são representados por nenhum termo em $\mathbb{P}$. Existe, de facto, uma fórmula que não é verificada por nenhum processo: a fórmula que representa a falsidade (absoluta) e que é constituida pela constante ff. Assim, um processo inactivo é aquele em que toda a transição conduz a um processo que verifica ff, *i.e.*, na notação que estamos a introduzir, $[Act]\mathit{ff}$. Logo, um processo que verifique $[Act]\mathit{ff}$ não poderá realizar qualquer transição.

Dualmente, todo o processo verifica tt, a fórmula composta pela constante que representa a verdade (absoluta). Logo um processo que pode realizar pelo menos uma transição (etiquetada por qualquer acção em Act) satisfaz a fórmula $\langle Act\rangle tt$. De facto, o conjunto dos processos que verificam $\langle Act\rangle tt$ coincide com $\mathbb{P} - \{E |\ E \sim \mathbf{0}\}$, como esperaríamos. Mais uma vez, por convenção, é usual abreviar $\langle Act\rangle$ e $[Act]$ por $\langle -\rangle$ e $[-]$, respectivamente. Em geral, para qualquer $K \subseteq Act$, a expressão $-K$ como argumento de qualquer dos conectivos anteriores, abrevia $Act - K$ (logo, $-$ é abreviatura de $Act - \emptyset = Act$).

Conectivos Modais

Os conectivos de possibilidade e necessidade acima introduzidos permitem-nos qualificar propriedades cuja validade se afirma em estados da vida do processo alcançados pela realização de transições. Por isso mesmo são ditos *modais*, já que quando aplicados a uma fórmula ϕ, estabelecem o *modo* em que esta é verdadeira.

Suponhamos que introduzimos estes conectivos numa linguagem proposicional com os operadores booleanos usuais. A lógica obtida é dita *modal*. Um modelo para esta lógica é construido sobre o conjunto $\mathbb{P}$ dos processos (*i.e.*, dos termos na álgebra de processos que temos vindo a considerar) equipado com a família de relações de transição $\{\xrightarrow{x} |\ x \in Act\}$ definida pela semântica operacional da linguagem dos processos. A característica fundamental da lógica reside na *relatividade* da noção de verdade.

Recordemos que a noção de modelo em lógica proposicional clássica é baseada numa atribuição de valores lógicos às variáveis, *i.e.*, numa função $V : Var \longrightarrow \mathbf{2}$, onde Var é um conjunto de variáveis proposicionais. Isto significa que uma proposição é verdadeira ou falsa em termos absolutos: há um único referencial, determinado quando se fixa a atribuição V, no qual a validade de uma fórmula se define indutivamente sobre a sua estrutura (por exemplo, uma conjunção será verdadeira sse as suas parcelas o forem).

Aqui, e pelo contrário, a atribuição de valores às variáveis torna-se relativa aos estados da computação, exprimindo-se por uma função $V : Var \times \mathbb{P} \longrightarrow \mathbf{2}$ ou, equivalentemente, $V : Var \longrightarrow \mathcal{P}(\mathbb{P})$, *i.e.*, a cada variável associa-se o conjunto de estados (processos) em que o seu valor é fixado como tt.

[1] Recorde que, para um predicado $p(x)$, $\forall_x\ p(x)$ sse $\neg\exists_x\ \neg p(x)$.

Na próxima secção vamos definir rigorosamente esta lógica baseada nas duas modalidades referidas. Somos, porém, já agora capazes de traduzir numa fórmula a propriedade com que iniciamos esta digressão. Assim, ϕ_0 escrever-se-á como

$$\langle rec, alo \rangle tt$$

se rec e alo representarem as acções de recolha de um passageiro e de alocação a um serviço, respectivamente. Mas mais interessante será dizer que ϕ_0 se verifica após a saída da viatura para o trabalho diário, dito de outro modo,

$$[sai]\langle rec, alo \rangle tt$$

NOTA: As ***lógicas modais*** são particularmente adequadas para captar propriedades de sistemas em movimento, na medida em que nos permitem exprimir e discutir as relações entre o ***mundo*** actual, i.é, aquele em que se coloca o observador, e outros ***mundos*** possíveis, em relação com ele. Não surpreende, pois, a sua popularidade nas Ciências da Computação, ciências que estudam sistemas essencialmente dinâmicos.

Temos em mãos o exemplo dos processos. Aqui partimos de um conjunto de termos de uma álgebra (*e.g.* $\mathbf{0}$ ou $a.b.\mathbf{0} + a.d.f.\mathbf{0}$), representando estados, que, por sua vez se relacionam pela ocorrência de transições etiquetadas por elementos de Act. Outro exemplo é fornecido pelos sistemas de transição gerados pelos programas sequenciais, onde os estados são associações de endereços da memória a valores e as transições etiquetadas por programas. As lógicas modais adequadas a este caso designam-se por ***lógicas de programas*** ou ***dinâmicas***, entre as quais se contam a lógica de Floyd-Hoare [Hoa69] e o cálculo de ***weakest preconditions*** [Dij76] que foram possivelmente abordadas no estudo da programação imperativa. Note-se, de passagem, que no primeiro caso a riqueza estrutural está localizada no conjunto de estados (*cf.* a estrutura induzida pela álgebra de processos), enquanto, no segundo, reside nas etiquetas (onde se define uma álgebra de programas cujos termos são, por exemplo, '$p_1;p_2$', 'if b then p_1 else p_2' ou 'while b do p_1'). Em ambos, a noção computacional de ***estado*** é introduzida na lógica, num sentido matematicamente preciso, como patamar de interpretação das fórmulas.

A origem das lógicas modais é, porém, muito anterior e o seu campo de aplicabilidade muito mais vasto. Classicamente estas lógicas diferenciam afirmações que são necessariamente verdadeiras de outras que apenas o podem ser. Proposições sempre verdadeiras ou falsas são ditas, respectivamente, ***necessárias*** e ***impossíveis.*** Por seu lado, uma proposição ***contingente*** não será nem necessária nem impossível, enquanto se classificam de ***possíveis*** todas as proposições não impossíveis. O significado exacto desta classificação é definido de diversas formas. O quatro conceitos — ***necessidade, possibilidade, impossibilidade*** e ***contingência*** — são ditos ***modais*** porque em lógica medieval eram encarados como os ***modos*** segundo os quais se discutia a veracidade de uma proposição.

Os sistemas de transição referidos pertencem à classe das estruturas — ditas de ***Kripke*** — classicamente utilizadas para interpretação das lógicas modais. Estas são formadas por um conjunto W (de ***mundos***) sobre o qual se toma uma relação binária, dita de ***acessibilidade.*** De facto, em Matemática, as lógicas modais são utilizadas como o contexto mais adequado para descrever, classificar e raciocinar sobre ***estruturas relacionais*** genéricas (ver [BRV95] para uma introdução razoavelmente acessível).

No nosso caso, W é o conjunto dos termos gerados pela álgebra de processos considerada. Por outro lado, generalizamos a relação de acessibilidade a uma família de relações de transição etiquetadas.

4.2 Uma Lógica Modal para Processos

A Linguagem Modal

A discussão feita na secção anterior permite-nos introduzir a seguinte linguagem modal — que designaremos por $\mathbb{L}_M$ — para exprimir propriedades comportamentais dos processos.

$$\phi \ ::= \ tt \ | \ ff \ | \ \phi_1 \wedge \phi_2 \ | \ \phi_1 \vee \phi_2 \ | \ \langle K \rangle \phi \ | \ [K]\phi$$

onde $K \subseteq Act$.

A relação fundamental entre um processo E e uma fórmula ϕ é a *relação de satisfação*, que se representa por $E \models \phi$. Esta é definida indutivamente sobre a estrutura das fórmulas. Assim,

$$\begin{array}{lll}
E \models tt & & \\
E \not\models ff & & \\
E \models \phi_1 \wedge \phi_2 & \text{sse} & E \models \phi_1 \;\wedge\; E \models \phi_2 \\
E \models \phi_1 \vee \phi_2 & \text{sse} & E \models \phi_1 \;\vee\; E \models \phi_2 \\
E \models \langle K \rangle \phi & \text{sse} & \exists_{F \in \{E' \mid E \xrightarrow{a} E' \text{ e } a \in K\}} \, . \, F \models \phi \\
E \models [K] \phi & \text{sse} & \forall_{F \in \{E' \mid E \xrightarrow{a} E' \text{ e } a \in K\}} \, . \, F \models \phi
\end{array}$$

As propriedades dos processos que podemos exprimir através desta lógica são essencialmente propriedades relacionadas com a necessidade ou possibilidade imediatas de se realizarem determinadas acções. Vejamos alguns exemplos. Consideremos de novo a especificação de um controlador de exclusão mútua:

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

Podemos facilmente verificar os factos seguintes:

- $Sem \models \langle get \rangle tt$ que estabelece a possibilidade do processo Sem realizar get. De facto, $\exists_{F \in \{Sem' \mid Sem \xrightarrow{get} Sem'\}} \, . \, F \models tt$; basta tomar $F = put.Sem$.
- Contudo, $Sem \models [put]ff$, *i.e.* Sem não pode realizar a acção put. De facto, o conjunto de transições $T = \{Sem' \mid Sem \xrightarrow{put} Sem'\}$ é vazio pelo que a expressão $\forall_{F \in T} \, . \, F \models ff$ se torna vacuosamente verdadeira.
- A única acção inicialmente permitida ao processo S é a acção não observável τ que resulta da sincronização de um get do semáforo com $\overline{get}$ realizado por um dos processos por ele controlados. Podemos exprimir isto pela fórmula $S \models [Act - \{\tau\}]ff$, ou, abreviadamente, $\models [-\tau]ff$.
- Após a ocorrência de τ, contudo, S pode realizar qualquer dos eventos críticos $c_1, c_2, ..., c_i$, *i.e.*, $[\tau]\langle c_1, c_2, ..., c_i \rangle tt$.
- De forma análoga podemos concluir que após uma sincronização inicial com o semáforo (pela ocorrência conjunta de get em Sem e $\overline{get}$ no sub-processo P_j) e a realização do evento crítico c_j por P_j, a ocorrência de uma nova sincronização com o semáforo (agora via put e $\overline{put}$) é inevitável. Temos, portanto, $S \models [\tau][c_j](\langle - \rangle tt \wedge [-\tau]ff)$.

Como vimos no último exemplo a *inevitabilidade* da ocorrência de uma acção a é expressa pela fórmula $\langle - \rangle tt \wedge [-a]ff$. A primeira parcela da conjunção indica que existe pelo menos uma transição possível, enqunto a segunda estabelece que todas as transições, excepto as etiquetadas por a, conduzem a um estado que verifica ff. Como tal estado não existe, podemos concluir que, num processo que satisfaça a fórmula, apenas existem transições por a (e existem mesmo porque se força $\langle - \rangle tt$).

Vimos já que $\langle - \rangle tt$ exprime possibilidade de evolução (há pelo menos uma transição disponível) e que $[-]\mathit{ff}$ traduz terminação ou *deadlock*. Claramente, $\mathbf{0} \models [-]\mathit{ff}$. Curiosamente, as fórmulas duais — $\langle - \rangle \mathit{ff}$ e $[-]tt$ — são desinteressantes. A primeira é verificada por um processo E tal que

$$\exists_{F \in \{E' \mid E \xrightarrow{x} E' \wedge x \in Act\}} \, . \, F \models \mathit{ff}$$

Como nenhuma transição pode conduzir a um estado que não existe, $\langle - \rangle \mathit{ff}$ é equivalente à constante ff. Dualmente, a segunda fórmula equivale a tt. Por seu lado, a possibilidade de realização de uma sequência $a_1, a_2, a_3, ..., a_n$ de acções é captada na fórmula $\langle a_1 \rangle \langle a_2 \rangle \langle a_3 \rangle ... \langle a_n \rangle tt$. Notemos, por fim, que a verificação de que um processo satisfaz uma determinada propriedade modal se faz muito simplesmente por aplicação da definição de $\models$. Em particular, não é necessário calcular o grafo de transições do processo.

Equivalência Modal e Bissimulação

A relação de satisfação em $\mathbb{L}_M$ permite-nos definir um nova equivalência entre processos — a que resulta da satisfação do mesmo conjunto de fórmulas. De facto, para qualquer conjunto Γ, finito ou infinito, de fórmulas podemos definir a seguinte equivalência em $\mathbb{P}$:

$$E \equiv_\Gamma F \quad \text{sse} \quad \forall_{\phi \in \Gamma} \, . \, E \models \phi \;\text{ sse }\; F \models \phi$$

Se, por exemplo, Γ fôr o conjunto de todas as fórmulas do tipo $\langle x_1 \rangle \langle x_2 \rangle ... \langle x_n \rangle tt$, para cada $x_i \in Act$, a equivalência $\equiv_\Gamma$ relacionará os processos capazes de realizarem as mesmas sequências de acções. Neste caso os processos $a.b.\mathbf{0} + a.c.\mathbf{0}$ e $a.(b.\mathbf{0} + c.\mathbf{0})$, que, como bem sabemos, não são bissimilares, são identificados por esta equivalência. Já se Γ se restringisse às fórmulas do tipo $\langle x_1 \rangle \langle x_2 \rangle \langle x_3 \rangle ... \langle x_n \rangle [-]\mathit{ff}$, para cada $x_i \in Act$, estaríamos a considerar equivalentes os processos que realizam as mesmas sequências *completas* de acções, *i.e.*, aquelas que conduzem à "morte" (terminação) do seu comportamento. Os processos poderiam divergir nas restantes sequências de acções.

Dizemos, em particular, que os processos E e F são *modalmente equivalentes*, e escrevemos $E \equiv F$ quando $E \equiv_\Gamma F$ e Γ é o conjunto de *todas* as fórmulas bem formadas da lógica.

Uma questão muito importante é saber de que forma a equivalência modal se relaciona com a bissimulação (estrita) sobre a qual construimos o nosso cálculo de processos. O resultado fundamental é o seguinte

Lema 19 *Dados dois processos E e F, se $E \sim F$ então $E \equiv F$.*

O converso deste lema não é verdadeiro no caso geral. Por exemplo, considerem-se os processos $A \triangleq \sum_{i \geq 0} A_i$, onde $A_0 \triangleq \mathbf{0}$ e $A_{i+1} \triangleq a.A_i$, e $A' \triangleq A + \underline{fix}\ (X = a.X)$. Claramente, A e A' verificam as mesmas propriedades modais. No entanto $A \nsim A'$ porque, caso contrário, a transição $A' \xrightarrow{a} \underline{fix}\ (X = a.X)$ deveria ser correspondida por uma transição de A por a para um processo bissimilar a $\underline{fix}\ (X = a.X)$. Ora A apenas pode transitar para um A_j, para algum $j \geq 0$, e, obviamente, $A_j \nsim \underline{fix}\ (X = a.X)$.

O resultado é contudo válido para uma classe restrita, embora na prática suficientemente ampla, de processos. Assim,

Lema 20 *Dados dois processos E e F com imagem finita, se $E \equiv F$ então $E \sim F$.*

Para um processo E e uma acção $x \in Act$, define-se o conjunto $\{E' \mid E \xrightarrow{x} E'\}$ dito dos sucessores de E por x. Sempre que os elementos do conjunto $\{F \mid \exists_{\omega \in Act^*} \, . \, E \xrightarrow{\omega} F\}$ têm conjuntos finitos de sucessores para todo o $x \in Act$, E é dito um processo com *imagem finita*. Não é difícil verificar que os processos discutidos no último exemplo não estão nestas condições.

NOTA: Os resultados referidos sobre a relação entre a equivalência modal e a bissimulação estrita foram originalmente provados em [HM85] onde se considera uma lógica modal similar à que aqui introduzimos mas em que os conectivos modais são parametrizados por acções singulares (e não por conjuntos de acções). Tal lógica é conhecida como a lógica de Hennessy-Milner e constitui um ingrediente clássico da teoria dos processos (o capítulo 10 de [Mil89] é-lhe dedicado). Existem, porém, outro tipo de correspondências entre a linguagem das relações (em que se insere a definição de bissimulação) e as lógicas modais. Possivelmente o resultado mais geral é o teorema da bissimulação provado por De Rikje em [Rik93] que, em termos caseiros, afirma que a equivalência modal coincide com a bissimulação "algures noutro lado" (em rigor, em determinadas ultra-potências dos sistemas de transição gerados pelos estados alcançáveis a partir dos processos em causa). Por agora, porém, consideremos as provas dos lemas que enunciamos. No primeiro caso a prova é feita por indução sobre a estrutura das fórmulas. Assim,

Prova: Suponhamos que $E \sim F$ e tentemos mostrar que $E \models \phi$ sse $F \models \phi$ para toda a fórmula ϕ. Os casos de base — $\phi = tt$ ou $\phi = ff$ — são triviais. Quando ϕ é uma conjunção ou uma disjunção, o resultado decorre imediatamente da hipótese de indução. Vejamos, pois, os casos em que o conectivo principal de ϕ é uma modalidade, que são os verdadeiramente interessantes.

Seja, então, $\phi = \langle K \rangle \psi$, com $K \subseteq Act$. Se $E \models \phi$ é porque existe um $E' \in \mathbb{P}$ tal que $E \xrightarrow{a} E'$, para algum $a \in K$, e $E' \models \psi$. Como $E \sim F$, sabemos que existe uma transição $F \xrightarrow{a} F'$ para um $F' \in \mathbb{P}$ tal que $E' \sim F'$. Como, por hipótese de indução $E' \equiv F'$, temos que $F' \models \psi$. Então, de novo pela definição do conectivo $\langle K \rangle$, concluimos que $F \models \langle K \rangle \psi$. O argumento é similar para o caso $\phi = [K]\psi$.

❏

Vejamos, agora, a demonstração do resultado converso quando os processos em causa têm imagens finitas. Notemos que essa condição significa que todos os estados alcançáveis a partir do processo em causa têm um número finito de sucessores. Assim,

Prova: Suponhamos que $E \equiv F$ e procuremos encontrar uma bissimulação que os contenha. A candidata óbvia é a própria equivalência modal $\equiv$. Verifiquemos, pois, se esta é capaz de satisfazer as duas claúsulas definidoras de uma bissimulação. Como bem sabemos essas claúsulas impõem condições simétricas, pelo que o argumento a utilizar na verificação de uma delas é também (simetricamente) válido para a segunda. O que é óptimo porque nos reduz a prova a metade!

Assim, suponhamos que $E \xrightarrow{a} E'$, para algum $a \in Act$, e que $E \equiv F$. Tentemos contrariar o resultado formulando a hipótese de que não existe nenhum $F' \in \mathbb{P}$ tal que $F \xrightarrow{a} F'$ e $E' \equiv F'$.

Seja $S = \{G | \ F \xrightarrow{a} G\}$. O conjunto S é necessariamente não vazio, porque, se fosse vazio, teríamos $F \models [a]ff$ ao passo que $E \models \langle a \rangle tt$ uma vez que estamos a supôr a existência de uma transição de E por a. Nesse caso não seria válida a equivalência $E \equiv F$, que constitui a hipótese do lema. Notemos, porém, que S é um conjunto *finito* $\{G_1, G_2, ..., G_n\}$, dado que F tem imagem finita.

Se, de facto, se não tem $E' \equiv F'$, temos que, para cada $G_i \in S$, existe uma fórmula ϕ_i tal que $E' \models \phi_i$ mas $G_i \not\models \phi_i$. Segue-se, pois, que $E \models \langle a \rangle (\phi_1 \wedge \phi_2 \wedge ... \wedge \phi_n)$, o mesmo não sendo válido para F. Mas então, E não é modalmente equivalente a F, o que contradiz a hipótese. O resultado segue, pois, por contra-recíproco. Notemos, contudo, que o argumento utilizado até aqui só prova o lema porque permanece válido quando, em vez de E e F, começamos por considerar dois quaiquer processos no espaço dos processos alcançáveis a partir de E e de F, respectivamente. Da condição do lema decorre que qualquer desses processos tem um conjunto finito de sucessores para todo o $a \in Act$.

❏

Repare o leitor em que ponto desta última prova foi usada a condição de imagem finita. Notemos que se F tivesse um número não finito de sucessores iríamos necessitar de uma conjunção infinita na definição de ψ. A lógica que introduzimos é finitária pelo que tal conjunção não faz sentido. Se eliminarmos essa restrição da lógica, o que diminui a sua operacionalidade, podemos igualmente eliminar a condição de imagem finita sobre os processos chegando a um resultado mais geral, mas (computacionalmente) menos relevante.

4.3 Lógica Modal e Transições Observáveis

Os conectivos modais que introduzimos exploram a estrutura das transições em $\mathbb{P}$ do tipo $\xrightarrow{x}$, para $x \in Act$, relativizando a validade das fórmulas a conjuntos de estados alcançados através de determinadas transições. Sobre $\mathbb{P}$, porém, é possível definir outras famílias de relações que são computacionalmente relevantes. Uma delas é a das relações de ***transição observável***, $\overset{a}{\Longrightarrow} \subseteq \mathbb{P} \times \mathbb{P}$, em que as etiquetas são elementos de $L \cup \{\epsilon\}$. Recorde-se que uma $\overset{\epsilon}{\Longrightarrow}$-transição corresponde a zero ou mais transições elementares não observáveis (*i.e.* etiquetadas por τ).

Vamos, pois, começar por introduzir em $\mathbb{L}_M$ dois novos conectivos modais que exprimem, respectivamente, a possibilidade e a necessidade de uma propriedade ser válida após a realização de uma quantidade arbitrária de comportamento não observável. Para definir estes operadores consideremos mais dois casos na definição da relação de satisfação $\models$.

$$
\begin{array}{lll}
E \models \langle\!\langle\,\rangle\!\rangle\phi & \text{sse} & \exists_{F \in \{E' \mid E \overset{\epsilon}{\Longrightarrow} E'\}} \,.\, F \models \phi \\
E \models [\![\,]\!]\phi & \text{sse} & \forall_{F \in \{E' \mid E \overset{\epsilon}{\Longrightarrow} E'\}} \,.\, F \models \phi
\end{array}
$$

Por abreviatura podemos agora definir as "versões observáveis" de $\langle K \rangle$ e $[K]$, para $K \subseteq L$. Assim,

$$
\begin{array}{l}
\langle\!\langle K \rangle\!\rangle\phi \overset{\text{abv}}{=} \langle\!\langle\,\rangle\!\rangle \; \langle K \rangle \; \langle\!\langle\,\rangle\!\rangle\phi \\
[\![K]\!]\phi \overset{\text{abv}}{=} [\![\,]\!] \; [K] \; [\![\,]\!]\phi
\end{array}
$$

Vejamos alguns exemplos de propriedades exprimíveis à custa deste conectivos.

- $\langle\!\langle a \rangle\!\rangle\, tt$ representa a capacidade de realização observável da acção a. Por exemplo, o processo S discutido anterior (que simulava uma situação de exclusão mútua) satisfaz a fórmula $\langle\!\langle c_j \rangle\!\rangle\, tt$, indicando que o processo P_j pode realizar a acção crítica respectiva.
- Similarmente a fórmula $\langle\!\langle a_1 \rangle\!\rangle\langle\!\langle a_2 \rangle\!\rangle\langle\!\langle a_3 \rangle\!\rangle ... \langle\!\langle a_n \rangle\!\rangle\, tt$ exprime a possibilidade de realização da sequência observável de acções $a_1, a_2, a_3, ..., a_n$.
- Por seu lado $[\![-]\!]\mathit{ff}$ traduz a impossibilidade de realização de comportamento observável. A propriedade é satisfeita, não apenas por $\mathbf{0}$ (que de resto verifica também $[-]\mathit{ff}$), mas igualmente por processos como $D \triangleq \tau.D + \mathbf{0}$ ou $(\underline{fix}\ (X = a.X) \mid \underline{fix}\ (X = \overline{a}.X))\backslash_a$ que se envolvem numa cadeia interminável de acções invisíveis.

Um caso interessante é o da propriedade que estabelece a inevitabilidade da ocorrência de a, mas agora de um ponto de vista observacional. A inevitabilidade foi já expressa como $\langle - \rangle tt \wedge [-a]\mathit{ff}$. Reparemos, contudo, que a versão observacional desta fórmula — $\langle\!\langle - \rangle\!\rangle\, tt \wedge [\![-a]\!]\mathit{ff}$ — é demasiado fraca. A fórmula é, por exemplo, satisfeita pelo processo $P \triangleq a.P + \tau.\mathbf{0}$, apesar de haver a possibilidade em P de a acabar por nunca ocorrer. Podemos fortalecê-la forçando a possibilidade de transições observáveis ocorrerem após a realização de comportamento invisível, *i.e.*, $[\![\,]\!]\,\langle\!\langle - \rangle\!\rangle\, tt \wedge [\![-a]\!]\mathit{ff}$. Esta fórmula já não é satisfeita por P, uma vez que $P \overset{\epsilon}{\Longrightarrow} \mathbf{0}$ e $\mathbf{0} \not\models \langle\!\langle - \rangle\!\rangle\, tt$. O problema permanece, porém, com o processo $P \triangleq a.P + \tau.P$, que, podendo igualmente nunca realizar a, satisfaz $[\![\,]\!]\,\langle\!\langle - \rangle\!\rangle\, tt$.

O problema reside na quantidade de movimento não observável admissível na definição dos novos conectivos modais. Se desejarmos exprimir que uma acção ocorre no termo da actividade interna do processo devemos limitar finitamente essa actividade. Dito de outro modo, temos de garantir que o processo é *convergente*, *i.e.*, incapaz de se comprometer num ciclo perpétuo de actividade interna não observável. Para isso vamos considerar um novo conectivo modal similar a $[\![\ ]\!]$, mas com informação adicional sobre convergência.

$$E \models [\![\downarrow]\!]\phi \qquad \text{sse} \qquad E\downarrow \wedge \forall_{F \in \{E' \mid E \stackrel{\epsilon}{\Longrightarrow} E'\}} \,.\, F \models \phi$$

onde $E\downarrow$ significa que o processo E é convergente.

Podemos, agora, definir uma fórmula para a inevitabilidade que exclua a possibilidade de divergência:

$$[\![\downarrow]\!]\langle\!\langle - \rangle\!\rangle tt \wedge [\![-a]\!]\mathit{ff}$$

Diversas combinações destes conectivos revelam-se úteis na especificação de processos. Por exemplo, podemos definir por abreviatura operadores que forcem a convergência do comportamento interno até determinadas transições ocorrerem ou uma dada fórmula ser satisfeita. É o caso de $[\![\downarrow K]\!] \stackrel{\text{abv}}{=} [\![\downarrow]\!][K][\![\ ]\!]$, $[\![K\downarrow]\!] \stackrel{\text{abv}}{=} [\![\ ]\!][K][\![\downarrow]\!]$ ou mesmo $[\![\downarrow K \downarrow]\!] \stackrel{\text{abv}}{=} [\![\downarrow]\!][K][\![\downarrow]\!]$. A semântica destes conectivos derivados é a que se esperaria. Por exemplo,

$$E \models [\![\downarrow K]\!]\phi \qquad \text{sse} \qquad E\downarrow \wedge \forall_{F \in \{E' \mid E \stackrel{a}{\Longrightarrow} E' \wedge a \in K\}} \,.\, F \models \phi$$

O leitor deverá verificar que, apesar de $[-]tt$ ser equivalente a tt, e, por isso válida em qualquer processo, apenas os processo convergentes satisfazem $[\![\downarrow -]\!]tt$. Assim, a fórmula $[\![\downarrow a_1 \downarrow]\!][\![\downarrow a_2 \downarrow]\!]...[\![\downarrow a_n \downarrow]\!]tt$ exprime a realização convergente de uma sequência $a_1 a_2 ... a_n$ de acções observáveis.

4.4 Propriedades Temporais e Computações

Transições *vs* Computações

Ao longo deste curso temos sublinhado a ideia de que os *sistemas de transição* constituem uma ferramenta adequada para a definição da semântica dos processos (e, mais genericamente, dos programas e dos sistemas computacionais). Vimos, também, que sobre eles podemos interpretar lógicas — a que chamamos *modais* — que relativizam a validade das proposições aos estados que os processos vão alcançando pela realização de transições. Vimos, por fim, como estas são capazes de exprimir propriedades *locais* dos processos, nomeadamente acerca da possibilidade ou da necessidade de ocorrência de determinadas acções, que encaramos como transformadores de estados.

Na especificação de sistemas concorrentes deparamos, no entanto, com a necessidade de formular propriedades, não apenas sobre transições particulares, mas também sobre a evolução global dos processos, *i.e.* os padrões comportamentais que estes podem ou devem exibir.

Sabemos, por exemplo, representar modalmente o facto de uma viatura, no exemplo da Rede de Táxis, poder regressar à Central, *i.e.*

$$\phi_1 \;=\; \langle reg \rangle tt$$

A possibilidade deste regresso não é contudo válida em todos os estados do processo que modela a viatura. Em particular, na especificação que consideramos (*cf.*, exercício I.20), não se verifica no estado inicial do processo (no qual a viatura se encontra, precisamente, na garagem da Central). O que, em rigor, gostaríamos de exprimir é o facto de, ao longo da sua vida, a viatura acabar por atingir um (ou mais) estado(s) em que o regresso é uma possibilidade efectiva. Dito de outra forma, ϕ_1 verifica-se ou, então, existe pelo menos uma transição que conduz a um estado em que ϕ_1 se verifica ou, então, existe pelo menos uma transição que ... O que seríamos tentados a escrever como

$$\phi_2 \;=\; \langle reg\rangle tt \vee \langle -\rangle\langle reg\rangle tt \vee \langle -\rangle\langle -\rangle\langle reg\rangle tt \vee \langle -\rangle\langle -\rangle\langle -\rangle\langle reg\rangle tt \vee \; ...$$

Não podendo, sem comprometer o seu significado, substituir ϕ_2 por uma disjunção finita, deparamos com uma fórmula que não é bem formada na lógica que atrás introduzimos. Vamos, pois, tentar formalizar o significado que intuitivamente atribuímos às reticências.

O ponto fundamental reside no facto de estarmos, agora, interessados em propriedades, não sobre as transições individualmente consideradas, mas sobre as *sequências de transições* que os processos realizam. Num processo E tais sequências constituem *caminhos*, ou cursos de evolução, ao longo do sistema de transição gerado por E.

Estamos particularmente interessados em *caminhos maximais*, *i.e.* nas sequências infinitas ou nas finitas que conduzem à terminação do processo, ou seja, a estados a partir dos quais não são possíveis quaisquer transições. No que se segue referir-nos-emos a estes caminhos como as *computações* dos processos.

Um outro exemplo da necessidade de raciocinar sobre as computações, e não apenas sobre as transições individuais, é fornecido pelos processos A e A' discutidos atrás. Aí definimos

$$\begin{aligned} A &\triangleq \sum_{i\geq 0} A_i \quad \text{ com } \; A_0 \triangleq \mathbf{0} \text{ e } A_{i+1} \triangleq a.A_i \\ A' &\triangleq A + \underline{fix}\;(X \;=\; a.X) \end{aligned}$$

e argumentamos que, apesar de $A \not\sim A'$, não havia nenhuma propriedade modal que os distinguisse. De facto, cada A_i falha a propriedade $\langle a\rangle^{i+1} tt$, que A' verifica. Mas teríamos, de novo, que recorrer a uma disjunção infinita para construir uma fórmula que qualquer A_i falhasse. O que, em verdade, distingue um processo do outro é a capacidade de A' realizar a acção a indefinidamente e essa é uma capacidade a "longo prazo", uma capacidade *temporal*.

Propriedades Temporais

Vamos designar por *propriedades temporais* aquelas que apenas faz sentido exprimir sobre as computações dos processos. Mas que propriedades são essas? Classicamente, distinguem-se dois tipos básicos:

- As propriedades de *segurança*, que se relacionam com a invariância de uma determinada característica comportamental (por exemplo, a ausência de *deadlock*). Intuitivamente podemos dizer que uma propriedade deste tipo estabelece que "as coisas indesejáveis não acontecem". No já muito batido exemplo do controlador de exclusão mútua, uma propriedade de segurança seria $\psi_1 =$ *A exclusão mútua não é violada*. No domínio da programação sequencial o típico requisito de segurança é a exigência de que o programa produza resultados correctos e é conhecido como a *correcção parcial* do programa.

- As propriedades de *animação* que estabelecem a necessidade ou a possibilidade de "coisas desejáveis acabarem por se verificar". Voltando ao exemplo anterior notemos que o requisito de segurança ψ_1 pode ser satisfeito por um controlador que permanentemente autorize apenas um dos processos a entrar na sua região crítica, ficando o outro incapaz de prosseguir. Ou até por um semáforo autista que impeça qualquer processo de aceder à respectiva região crítica. Em qualquer dos casos, mesmo que por omissão, se garante a exclusão mútua. Assim, a especificação do controlador não pode omitir uma propriedade que estabeleça que $\psi_2 =$ *cada processo é capaz, de eventualmente ter acesso à sua região crítica.* Deste modo se garante a ausência de *livelock* — ψ_2 é, pois, uma propriedade de animação. No caso sequencial a propriedade de animação que é forçoso garantir é a *terminação* da execução do programa. A dupla garantia da correcção parcial e da terminação é referida como a *correcção total* do programa. No caso da programação concorrente, porém, a ontologia da animação é muito mais rica. Temos, por exemplo, propriedades relacionadas com o progresso dos sistemas (*e.g.* a acção *reg* ocorre eventualmente numa computação do sistema), propriedades recorrentes (*e.g.* a acção *reg* ocorre um número infinito de vezes na vida do sistema), propriedades de persistência (*e.g.* a partir de um determinado estado, a acção *reg* está continuamente disponível), ...

A especificação de um sistema concorrente inclui, regra geral, uma componente de animação e uma componente de segurança. Além disso qualquer propriedade sobre o comportamento de um sistema pode ser expressa como uma combinação de uma propriedade de cada um dos dois tipos referidos.

NOTA: A divisão das propriedades temporais em dois grandes grupos — *segurança* e *animação* — foi originalmente proposta por L. Lamport em [Lam77], onde surge também a sua ilustração em termos da ocorrência de "coisas desejáveis" ou "indesejáveis". Se interpretarmos essas "coisas" como factos verificáveis (observáveis) em tempo finito, podemos dizer as propriedades temporais se distinguem pela frequência de ocorrências de "coisas desejáveis": sempre (segurança), pelo menos uma vez (progresso), um número infinito de vezes (recorrência), etc.

Desde [Lam77] têm sido propostas diversas formulações rigorosas destas propriedades [AS85, Lam83] e outros esquemas de classificação. Em particular, existem diversas tentativas de organizar hierarquicamente as propriedades de animação (*e.g.* [MP90]).

Note-se, porém, que para especificar de requisitos de animação se torna necessária uma linguagem capaz de definir propriedades sobre comportamentos *infinitos*, enquanto para a segurança é suficiente lidar com comportamentos *finitos*, ou com prefixos finitos de comportamentos infinitos. As técnicas de prova usadas na verificação destes requisitos são, igualmente, distintas. Assim, para estabelecer uma propriedade de segurança é suficiente mostrar que esta se verifica no estado inicial do processo e que é preservada pelas transições. Consequentemente, um argumento indutivo sobre o comprimento da computação estabelece que a propriedade é sempre válida. A indução aqui é implícita: surge na justificação do método de prova e não na sua aplicação. Na área dos métodos formais de desenvolvimento de programas (por exemplo, em [Jon86] ou [Oli90]) é comum depararmos com provas deste tipo, por exemplo, na demonstração da preservação do invariante pelas operações de um modelo (ou não fosse este um típico requisito de segurança). Já a prova de propriedades de animação recorre frequentemente a um argumento indutivo explícito sobre uma observação do conjunto de estados que mede a "distância" até à verificação da propriedade desejada.

4.5 Interpretação das Propriedades Temporais

Como Interpretar as Propriedades Temporais?

Vimos já que o que fundamentalmente distingue as propriedades temporais das propriedades modais anteriormente consideradas é o facto de as primeiras serem formuladas sobre o conjunto das *computações* dos processos. Assim, parece razoável supôr que as mesmas fórmulas modais, se interpretadas sobre sequências de transições elementares, poderiam ser "re-utilizadas" para exprimir propriedades temporais. Por exemplo, poderíamos tomar, como universo de interpretação, um sistema de transição cujos estados continuassem a ser os termos de $\mathbb{P}$, mas em que, na definição das transições, se tomasse $\xrightarrow{\omega} \subseteq \mathbb{P}\times\mathbb{P}$, com $\omega \in Act^*$ (em vez de $\xrightarrow{a} \subseteq \mathbb{P} \times \mathbb{P}$, com $a \in Act$. Poderíamos, ainda, ser um pouco mais ambiciosos e restringir esta relação às sequências que efectivamente representam computações válidas (de acordo com a definição acima).

Em qualquer caso, obtemos uma estrutura de interpretação muito mais rica que, por si só, sugere a introdução de novos operadores na lógica. Operadores que realizem uma "quantificação" sobre as computações, da mesma forma que $[-]$ e $\langle - \rangle$ "quantificam" sobre as transições elementares. As lógicas resultantes são classificadas como *lógicas temporais*.

De facto, transições etiquetadas por sequências de acções, ou por determinados tipos destas, introduzem o elemento de *temporalidade* que pretendemos dominar. Podemos, mesmo, *codificar* estas relações em estruturas relacionais que modelam discretamente o tempo como um conjunto de instantes e uma ordem parcial entre eles. Esta abordagem, com as suas múltiplas variantes, constitui um modo razoavelemente popular de analisar as propriedades temporais dos sistemas de computação, a partir do trabalho pioneiro de A. Pnueli [Pnu77] e outros.

NOTA: De facto, originalmente, as lógicas temporais definem-se como lógicas modais sobre sistemas de transição não etiquetados onde as relações de transição são simples ordens parciais. Se olharmos para essas ordens como a relação familiar de precedência temporal e o conjunto de estados como um conjunto de instantes, as duas famílias básicas de operadores modais colapsam em dois únicos conectivos (uma vez que omitimos a etiquetagem por acções). Temos, então, $\langle\ \rangle$ e $[\ ]$ que exprimem, respectivamente, *eventualidade* e *invariância*. Diferentes tipos de lógicas temporais são obtidas considerando diferentes maneiras de ordenar o tempo. A usual ordem total leva-nos aos modelos *lineares*. Se, porém, admitirmos não linearidade à esquerda ou à direita obtemos os modelos *ramificados* que permitem considerar distintos cenários futuros ou diferentes histórias passadas, respectivamente. Recorde-se que uma relação de ordem $\mathcal{R}$ é linear à direita se $(a\mathcal{R}c \wedge b\mathcal{R}c) \Rightarrow (a = b \vee a\mathcal{R}b \vee b\mathcal{R}a)$. Dualmente, diz-se linear à esquerda se $(c\mathcal{R}a \wedge c\mathcal{R}b) \Rightarrow (a = b \vee a\mathcal{R}b \vee b\mathcal{R}a)$. Existem inúmeras variantes desta aproximação e das suas aplicações às ciências da computação (ver, por exemplo, [Sti92]).

A abordagem que vamos seguir neste curso é, contudo, um pouco distinta. Assim, em vez de interpretarmos as fórmulas modais directamente sobre uma estrutura relacional mais geral, vamos encarar as propriedades temporais como *limites* de propriedades modais sobre transições elementares. Esta visão das propriedades tem uma analogia, que queremos explorar, com a visão das computações de um processo como *limites* das transições elementares. Por exemplo, a possibilidade de um processo realizar numa determinada computação a transição etiquetada por a, pode ser vista como o limite das possibilidades de, em cada estado dessa computação ser possível a realização de a.

Porém, para desenvolvermos esta perspectiva, necessitamos de voltar a reflectir sobre as propriedades modais que atrás introduzimos. O que motiva

Um Segundo Olhar Sobre a Lógica Modal

A cada fórmula modal ϕ podemos associar o conjunto dos processo que a verificam, *i.e.* $\{E \in \mathbb{P} |\ E \models \phi\}$, conjunto que representaremos por $\|\phi\|$. Notemos que ϕ e $\|\phi\|$ são duas maneiras distintas de nos referirmos à *mesma* propriedade, ainda que ϕ o faça de forma mais conveniente uma vez que, regra geral, $\|\phi\|$ não é um conjunto finito.

Uma qualquer propriedade ϕ divide, pois, $\mathbb{P}$ em dois subconjuntos disjuntos: $\|\phi\|$ e o seu complemento (*i.e.* $\mathbb{P} - \|\phi\|$). Este último corresponderia a $\|\neg\phi\|$, se tivessemos introduzido na lógica o conectivo usual de negação ($\neg$). Mesmo não o tendo feito, podemos sempre referirmo-nos ao *complemento* de ϕ, que representaremos por ϕ^c, como a fórmula correspondente à propriedade $\mathbb{P} - \|\phi\|$. De facto, esta noção de fórmula complementar está presente na lógica mesmo sem a introdução explícita da negação. Podemos defini-la estruturalmente. Assim,

$$\begin{aligned} tt^c =& ff \\ ff^c =& tt \\ (\phi_1 \wedge \phi_2)^c =& \phi_1^c \vee \phi_2^c \\ (\phi_1 \vee \phi_2)^c =& \phi_1^c \wedge \phi_2^c \\ (\langle a \rangle \phi)^c =& [a]\phi^c \end{aligned}$$

Esta correspondência entre fórmulas modais e conjuntos de processos fornece-nos uma definição alternativa da semântica da lógica que temos vindo a considerar. Para isso, porém, torna-se necessário definir directamente os conjuntos $\|\phi\|$, por indução sobre a estrutura das fórmulas, e não em termos da relação $\models$. É fácil concluir que

$$\begin{aligned} \|tt\| =& \mathbb{P} \\ \|ff\| =& \emptyset \\ \|\phi_1 \wedge \phi_2\| =& \|\phi_1\| \cap \|\phi_2\| \\ \|\phi_1 \vee \phi_2\| =& \|\phi_1\| \cup \|\phi_2\| \end{aligned}$$

As propriedades modais, vistas como conjuntos de processos, formam uma álgebra booleana (basta verificar que são fechadas para as operações de intersecção e complementação e que satisfazem os axiomas booleanos usuais).

Mas como vamos interpretar $[K]\phi$ ou $\langle K \rangle \phi$? Tal como a conjunção é interpretada por uma função (binária) que a dois conjuntos de processos (por exemplo, $\|\phi_1\|$ e $\|\phi_2\|$) faz corresponder a sua intersecção, também os conectivos modais podem ser vistos como funções (unárias) entre conjuntos de processos. Vamos designar essas funções por $\|[K]\|$ e $\|\langle K \rangle\|$, respectivamente. Por exemplo, $\|[K]\|$ faz corresponder a um conjunto X de processos um outro conjunto de processos cujas derivadas imediatas por acções em K constituem X. Formalmente,

$$\|[K]\| \ = \ \lambda_{X \subseteq \mathbb{P}} \ . \ \{F \in \mathbb{P} |\ \text{se } F \xrightarrow{a} F' \text{ e } a \in K \ \ \text{então} \ \ F' \in X\}$$

De forma similar se define $\|\langle K \rangle\|$,

$$\|\langle K \rangle\| \ = \ \lambda_{X \subseteq \mathbb{P}} \ . \ \{F \in \mathbb{P} |\ \exists_{F' \in X, a \in K} \ . \ F \xrightarrow{a} F'\}$$

Note-se que o efeito destas funções é operar, em cada processo em X, uma redução ao "estado anterior" indexada pelas acções em K. Tal como acontece com as interpretações dos conectivos booleanos usuais, estas são igualmente funções *monótonas* relativamente à inclusão de conjuntos, *i.e.*

$$\text{Se } X_1 \subseteq X_2 \quad \text{então} \quad \|[K]\|(X_1) \subseteq \|[K]\|(X_2) \;\; \text{e} \;\; \|\langle K\rangle\|(X_1) \subseteq \|\langle K\rangle\|(X_2)$$

Além disso não é difícil mostrar que as funções $\|[K]\|$ e $\|\langle K\rangle\|$ verificam as seguintes igualdades:

$$\begin{aligned}
\|[K]\|(\mathbb{P}) &= \mathbb{P} \\
\|\langle K\rangle\|(\emptyset) &= \emptyset \\
\|[K]\|(X_1 \cap X_2) &= \|[K]\|(X_1) \cap \|[K]\|(X_2) \\
\|\langle K\rangle\|(X_1 \cup X_2) &= \|\langle K\rangle\|(X_1) \cup \|\langle K\rangle\|(X_2)
\end{aligned}$$

Podemos, agora, completar a tabela anterior, de interpretação das fórmulas modais, definido,

$$\begin{aligned}
\|[K]\phi\| &= \|[K]\|(\|\phi\|) \\
\|\langle K\rangle\phi\| &= \|\langle K\rangle\|(\|\phi\|)
\end{aligned}$$

NOTA: Dissemos acima que a correspondência entre fórmulas e conjuntos de processos fornecia uma definição alternativa da semântica da lógica modal. Notemos, porém, que, em rigor, a definição estrutural de $\| \ \|$ para o caso dos conectivos modais só é válida assumindo que trabalhamos num universo de processos fechado para a relação de transição, o que é o caso de $\mathbb{P}$.

Feito este último comentário, o que é interessante é que obtivemos um novo *modelo* para a lógica de sabor mais algébrico, em alternativa ao que já tinhamos (*i.e.* os sistemas de transição ou, genericamente, as estruturas relacionais). De facto, e seguindo um procedimento usual em Matemática, podemos olhar estes conjuntos de processos e operadores sobre eles (*e.g.* $\cap$, $\emptyset$, $\|[K]\|$, ...) como objectos e operadores abstractos que respeitam determinados axiomas equacionais. Podemos, assim, definir a noção de *álgebra modal*:

Definição. Uma álgebra modal é um tuplo $(P, 1, \sqcap, \neg, \{\|[x]\| \mid x \in Act\})$ onde $(P, 1, \sqcap, \neg)$ forma uma álgebra booleana e, para cada $x \in Act$, o operador $\|[x]\| : P \longrightarrow P$ verifica $\|[x]\|(1) = 1$ e $\|[x]\|(m \sqcap n) = \|[x]\|(m) \sqcap \|[x]\|(n)$.

Note-se que os restantes operadores — 0, $\sqcup$ e $\|\langle x\rangle\|$ — são definidos por abreviatura; em particular $\sqcup \overset{\text{abv}}{=} \neg\sqcap\neg$ e $\|\langle x\rangle\| \overset{\text{abv}}{=} \neg\|[x]\|\neg$.

Esta linha de raciocínio é ilustrativa de uma estragégia mais geral, e muito bem sucedida, em Lógica. Trata-se de traduzir conceitos e problemas da lógica em termos algébricos, aplicar, de seguida, os métodos da Álgebra Universal na sua resolução e traduzir a solução de novo para a linguagem da Lógica. Esta estratégia corresponde a uma disciplina própria — a *Lógica Algébrica* — que se iniciou com De Morgan, no final do século XIX, quando a procura de estruturas correspondentes à lógica proposicional clássica conduziu à formulação das familiares álgebras booleanas. As álgebras modais, que acabamos de introduzir, desempenham um papel análogo relativamente à lógica modal [BRV95].

Historicamente, as álgebras modais constituiram um dos primeiros modelos para a lógica modal. De facto, a ponte entre ambas surgiu num artigo, hoje clássico, de Jónsson e Tarski em 1951 [JT51], cerca de uma década antes do desenvolvimento da semântica relacional que temos vindo a adoptar, por Kripke e outros [Kri63].

Tal como os sistemas de transição, as álgebras modais podem ser classificadas de acordo com certas condições adicionais que verificam. No caso dos sistemas de transição estas surgem como propriedades da família de relações que os definem (*e.g.* simetria, serialidade, etc.). Nas álgebras modais tais condições tomam usualmente a forma de um conjunto de equações — um exemplo típico é o caso das *álgebras dinâmicas* de V. Pratt [Pra92] que constituem a contrapartida algébrica de certas lógicas (modais) dos programas imperativos.

A correspondência entre estes dois modelos semânticos (relacional e algébrico) é muito útil quando queremos raciocinar sobre propriedades exprimíveis neste tipo de lógicas. Recordemos que esta nossa incursão algébrica se tornou necessária para tentar definir o significado das propriedades temporais dos processos. De facto, cada sistema de transição determina uma álgebra modal sobre a potência do respectivo conjunto de estados, como vimos, de forma particular, para o caso do sistema $(\mathbb{P}, \{\xrightarrow{x} \mid x \in Act\})$. Dualmente, as álgebras modais determinam sistemas de transição, segundo uma construção similar à do teorema da representação das álgebras booleanas [Sto36]. Como em tantos outros campos das ciências da computação, estamos aqui perante mais uma manifestação de uma dualidade de Stone.

Propriedades Temporais Como Limites

Como dissemos, pretendemos ver as propriedades temporais como *limites* de propriedades modais sobre as transições elementares dos processos. Esta visão das propriedades tem uma analogia com a semântica operacional dos processos. De facto, as computações, que constituem o alvo deste tipo de propriedades, resultam do encadeamento, potencialmente não finito, dessas transições.

Raciocinemos, pois, por analogia. Quando queremos exprimir padrões comportamentais complexos, e também eles potencialmente não finitos, o mecanismo que utilizamos é a recursão sobre $\mathbb{P}$, introduzida quer através da igualdade definicional ($\triangleq$) quer por um operador específico ($\underline{fix}$). Que sentido fará introduzir o mesmo mecanismo nas fórmulas modais? Certamente escrever

$$X \triangleq \langle a \rangle tt$$

não levanta qualquer dificuldade: X é apenas uma fórmula verificável exactamente pelos mesmos processos que verificam $\langle a \rangle tt$. Mas que diremos, por exemplo, de

$$X \triangleq \langle a \rangle X$$

Sabemos que uma maneira de interpretar as fórmulas modais é associa-las aos conjuntos de processos que as satisfazem. Em particular, a interpretação de $\langle a \rangle X$ é uma função em $\mathcal{P}(\mathbb{P})$ que ao conjunto de processos que X representa, qualquer que ele seja, faz corresponder o conjunto $\|\langle a \rangle\|(X)$. Assim, podemos interpretar a fórmula $X \triangleq \langle a \rangle X$ como uma *equação* em $\mathcal{P}(\mathbb{P})$ cujas soluções são os pontos fixos da função

$$\lambda_{X \subseteq \mathbb{P}} \,.\, \|\langle a \rangle\|(X)$$

Note-se que, por um ligeiro abuso de notação, identificamos a variável lógica X com uma variável sobre $\mathbb{P}$ (em rigor, deveríamos ter escrito $\|X\|$ na função acima).

A questão interessante é, agora, saber como caracterizar os pontos fixos da função em causa. No caso da definição dos processos o problema está muito simplificado na medida em que, como mostramos, um conjunto muito significativo de equações recursivas em $\mathbb{P}$ têm soluções únicas. O mesmo não se passa neste caso e, consequentemente, precisamos de saber identificar com clareza pelo menos alguns dos pontos fixos da equação, *i.e.* os subconjuntos $S \subseteq \mathbb{P}$ que verificam

$$S = \|\langle a \rangle\|(S)$$

A resolução deste tipo de equações não levanta problemas de maior. Contudo, o domínio em que as temos de resolver (*i.e.* o espaço $\mathcal{P}(\mathbb{P})$) é consideravelmente menos rico em propriedades algébricas que, por exemplo, o corpo dos reais. Não disporemos, portanto, de algoritmos expeditos comparáveis.

Vimos atrás que a função $\lambda_{X \subseteq \mathbb{P}} \, . \, \|\langle a \rangle\|(X)$ era monótona relativamente à inclusão de conjuntos, propriedade que, de resto, é partilhada pela interpretação em $\mathcal{P}(\mathbb{P})$ dos outros conectivos modais. Isto basta para podermos, no cálculo das soluções que procuramos, recorrer a um resultado clássico sobre a existência de pontos fixos de funções monótonas em recticulados completos — o *teorema de Knaster-Tarski* (ver Nota).

Instanciando este resultado para o recticulado $(\mathcal{P}(\mathbb{P}), \subseteq)$, concluimos que o *menor* dos pontos fixos de uma função monótona pode ser calculado como a intersecção generalizada (*i.e.* o ínfimo no recticulado em causa) do conjunto dos respectivos pré-pontos fixos. Estes, por sua vez, são aqueles elementos S de $\mathcal{P}(\mathbb{P})$ tais que $f(S) \subseteq S$. Dito de outro modo, e para o caso que nos interessa, são aqueles conjuntos de processos que verificam

$$\|\langle a \rangle\|(S) \ \subseteq \ S$$

Ora, conhecendo nós a definição do operador $\|\langle a \rangle\|$, podemos re-escrever a condição acima sob a forma de um predicado sobre os processos que fazem parte desses pré-pontos fixos. Diremos, assim, que S é um pré-ponto fixo de $\lambda_{X \subseteq \mathbb{P}} \, . \, \|\langle a \rangle\|(X)$ sse

$$\text{(PRE)} \qquad \text{Se } \ E \in \mathbb{P} \ \wedge \ E' \in S \ \wedge \ E \xrightarrow{a} E' \ \text{ então } \ E \in S$$

Vejamos que conjuntos S satisfazem esta condição. Claramente, o conjunto $\{\underline{fix} \ (A = a.A)\}$ verifica-a. Mas o mesmo se passa com o conjunto vazio (de facto se a premissa é falsa podemos concluir o que quisermos acerca do consequente da implicação). Chegamos, deste modo, à nossa primeira decepção: a fórmula, que parecia tão promissora, equivale a uma fórmula que é satisfeita pelo conjunto vazio de processos, *i.e.* à fórmula *ff*!

Felizmente, porém, esta não é a única solução da equação em mãos. Por verificação directa encontramos outra — o conjunto $\{\underline{fix} \ (A = a.A)\}$. O teorema de Knaster-Tarski não nos diz como obter uma solução arbitrária para a equação (foi um "golpe de vista" que nos conduziu a $\{\underline{fix} \ (A = a.A)\}$ ...), mas, contudo, fornece um método para calcular o outro ponto fixo extremo, *i.e.* o *maior* ponto fixo. De acordo com o teorema, este é calculado pela reunião generalizada (*i.e.* o supremo no recticulado em causa) do conjunto dos post-pontos fixos da função, *i.e.* daqueles subconjuntos S de $\mathbb{P}$ tais que

$$S \ \subseteq \ \|\langle a \rangle\|(S)$$

De novo podemos traduzir esta condição sob a forma de um predicado (POST) de acordo com,

$$\text{(POST)} \qquad \text{Se } \ E \in S \ \text{ então } \ E \xrightarrow{a} E' \ \text{ para algum } \ E' \in S$$

Qual será o maior subconjunto de $\mathbb{P}$ que verifica esta condição? A condição afirma que, se um processo pertence a S, então pode realizar a acção a e o processo que resulta dessa realização pertence igualmente a S. Podemos, assim, concluir que o maior subconjunto de $\mathbb{P}$ que verifica esta condição é o conjunto dos processos que apresentam pelo menos uma computação infinita da forma

$$E \xrightarrow{a} E_1 \xrightarrow{a} E_3 \xrightarrow{a} E_3 \xrightarrow{a} \ldots$$

Conseguimos, pois, exprimir à custa do maior ponto fixo da equação modal, a seguinte propriedade temporal:

$\phi =$ *Em qualquer ponto da sua evolução, o processo tem sempre a possibilidade de realizar a acção a*

O Caso Geral

Como este exemplo ilustrou, propriedades temporais podem ser formuladas como pontos fixos de determinadas equações. Equações que, como vimos, correspondem à interpretação em $\mathcal{P}(\mathbb{P})$ de fórmulas do tipo $X \triangleq \phi$, sendo X uma variável proposicional e ϕ uma fórmula modal onde X ocorre livre. A forma geral da interpretação em $\mathcal{P}(\mathbb{P})$ destas equações é

$$X \;=\; f(X)$$

onde $=$ é a igualdade entre conjuntos e

$$f \;=\; \lambda_{X\subseteq\mathbb{P}}\,.\,\|\phi\|$$

Com auxílio do teorema de Knaster-Tarski podemos identificar a menor e a maior solução de uma equação deste tipo, ou seja o menor e o maior ponto fixo de f.

NOTA: Como atrás se referiu, para garantir a existência de pontos fixos extremos é necessária a monotonia de f relativamente a $\subseteq$, o que se pode verificar para a interpretação em $\mathcal{P}(\mathbb{P})$ de todos os conectivos de $\mathbb{L}_M$. Aqui reside, porém, a razão pela qual se omitiu de $\mathbb{L}_M$ o conectivo usual de negação. De facto, na sua presença nem todas as equações modais exibem pontos fixos extremos. Repare-se que, por exemplo em $X = \neg X$, f não é monótona. É possível, contudo, introduzir $\neg$ desde que se imponha que em qualquer equação, toda a ocorrência livre da variável X esteja guardada por um número par de negações.

Por vezes as duas soluções são coincidentes (logo, únicas), mas no caso geral são distintas e existem, não raro, outras soluções intermédias. Como não possuímos um processo sistemático de cálculo destas soluções intermédias, seremos forçados a viver sem elas. No entanto vamos frequentemente recorrer às soluções extremas. Temos, pois, necessidade de introduzir alguma notação para a elas nos referirmos. Assim, vamos representar por

$$\mu X\,.\,\phi$$

o menor ponto fixo de $f \;=\; \lambda_{X\subseteq\mathbb{P}}\,.\,\|\phi\|$ e por

$$\nu X\,.\,\phi$$

o maior ponto fixo. No exemplo em mãos, mostramos que $\mu X\,.\,\langle a\rangle tt$ é equivalente a $f\!f$, enquanto $\nu X\,.\,\langle a\rangle tt$ garante a existência de pelo menos uma computação infinita de as. De modo análogo, a fórmula $\nu X\,.\,\langle\tau\rangle tt$ caracteriza os processos que se podem comprometer numa sequência infinita de acções não observáveis, ou seja, os processos *divergentes*.

Se o conectivo modal fôr parametrizado por um conjunto de acções $K \subseteq Act$, o processo que verifique $\nu X\,.\,\langle K\rangle tt$ pode realizar uma computação infinita composta por uma sequência qualquer de acções em K. Em particular, fazendo K coincidir com o próprio Act, a fórmula $\nu X\,.\,\langle -\rangle tt$ representa os processos que possuem pelo menos uma computação (qualquer) infinita, *i.e.* que têm a possibilidade de exibir um comportamento perpétuo.

Vimos, também, que no cálculo destes pontos fixos é útil exprimir as condições $f(X) \subseteq X$ e $X \subseteq f(X)$ em termos dos elementos de X, *i.e.*

$$\begin{array}{ll}
\text{(PRE)} & \text{Se} \quad E \in \mathbb{P} \text{ e } E \in f(X) \quad \text{então} \quad E \in X \\
\text{(POST)} & \text{Se} \quad E \in X \quad \text{então} \quad E \in f(X)
\end{array}$$

respectivamente.

NOTA: O teorema de Knaster-Tarski, publicado originalmente em [Tar55], é um resultado clássico na teoria dos recticulados. Fornecemos aqui, para referência, o seu enunciado e demonstração.

Teorema. Seja $(U, \sqsubseteq)$ um recticulado completo e $f : U \longrightarrow U$ uma função monótona, *i.e.*, tal que se $x \sqsubseteq y$, então $f(x) \sqsubseteq f(y)$. Então

$$\begin{aligned} m &= \bigsqcap \{x \in U \mid f(x) \sqsubseteq x\} \\ M &= \bigsqcup \{x \in U \mid x \sqsubseteq f(x)\} \end{aligned}$$

constituem, respectivamente, o menor e o maior ponto fixo de f.

Prova: Comecemos por provar que m é o menor ponto fixo de f. Seja $X = \{x \in U \mid f(x) \sqsubseteq x\}$ e tomemos um qualquer $x \in X$. Claramente, $m \sqsubseteq x$ e, sendo f monótona, $f(m) \sqsubseteq f(x)$. Por outro lado $f(x) \sqsubseteq x$, uma vez que $x \in X$. Logo podemos concluir que, para todo o $x \in X$, $f(m) \sqsubseteq x$. Isto significa que m é o *menor* pré-ponto fixo de f. Em particular $f(m) \sqsubseteq m$, o que, de novo pela monotonia de f, nos leva a $f(f(m)) \sqsubseteq f(m)$. Daqui concluimos que $f(m) \in X$ e, portanto, $m \sqsubseteq f(m)$. Mas então, $f(m) = m$ como queríamos. A segunda parte do teorema decorre desta: basta notar que se f é monótona em $(U, \sqsubseteq)$, então é ainda monótona no recticulado completo $(U, \sqsupseteq)$ formado pela ordem inversa. Se M é o menor ponto fixo de f em $(U, \sqsupseteq)$, será o *maior* ponto fixo da mesma função f em $(U, \sqsubseteq)$.

❏

Exemplo: $X \triangleq \phi \vee \langle a \rangle X$

Vamos investigar novas equações modais, aplicando esta estratégia para tentar caracterizar outras propriedades temporais. Comecemos, pois, por considerar a equação

$$X \triangleq \phi \vee \langle a \rangle X$$

A função implícita é

$$f \triangleq \lambda_{X \subseteq \mathbb{P}} \,.\, \|\phi\| \cup \|\langle a \rangle\|(X)$$

Vamos desenvolver a condição que define os seus pré-pontos fixos. Assim,

$$\begin{aligned} &(\text{PRE}) \ \text{Se} \quad E \in \mathbb{P} \ \text{ e } \ E \in f(X) \quad \text{então} \quad E \in X \\ &\equiv \ \text{Se} \quad E \in \mathbb{P} \ \text{ e } \ E \in (\|\phi\| \cup \|\langle a \rangle\|(X)) \quad \text{então} \quad E \in X \\ &\equiv \ \text{Se} \quad E \in \mathbb{P} \ \text{ e } \ E \in \{F \mid F \models \phi\} \cup \{F \in \mathbb{P} \mid \exists_{F' \in X} \,.\, F \xrightarrow{a} F'\} \quad \text{então} \quad E \in X \\ &\equiv \ \text{Se} \quad E \in \mathbb{P} \ \text{ e } \ E \models \phi \vee \exists_{E' \in X} \,.\, E \xrightarrow{a} E' \quad \text{então} \quad E \in X \end{aligned}$$

O menor conjunto de processos que verifica esta condição é formado por processos que possuem pelo menos uma computação ao longo da qual são capazes de realizar a acção a *até* a propriedade ϕ se verificar. Note-se, porém, e aqui está um ponto fundamental, que existe mesmo, nessa computação, um estado em que ϕ se verifica. E isto porque, ao tomar a intersecção destes conjuntos, ficamos com aqueles que atingem o estado em que ϕ se verifica num número *finito* de passos.

Vejamos, agora, qual será o maior dos pontos fixos de f. Para isso vamos "desenrolar" a condição que caracteriza os respectivos post-pontos fixos.

$$
\begin{array}{rlll}
\text{(POST)} & \text{Se} \quad E \in X & \text{então} & E \in f(X) \\
\equiv & \text{Se} \quad E \in X & \text{então} & E \in (\|\phi\| \cup \|\langle a\rangle\|(X)) \\
\equiv & \text{Se} \quad E \in X & \text{então} & E \in \{F |\ F \models \phi\} \cup \{F \in X |\ \exists_{F' \in X} \ .\ F \xrightarrow{a} F'\} \\
\equiv & \text{Se} \quad E \in X & \text{então} & E \models \phi \vee \exists_{E' \in X} \ .\ E \xrightarrow{a} E'
\end{array}
$$

O maior ponto fixo é formado tomando a reunião de todos os post-pontos fixos. Isto significa, neste exemplo, que nele vamos incluir não apenas os processos que podem realizar a até à verificação de ϕ, mas também aqueles que mantêm a possibilidade de realizar a indefinidamente, sem nunca chegar a atingir um estado que verifique ϕ.

Em resumo, diremos que tanto $\mu X \,.\, \phi \ \vee \ \langle a\rangle X$ como $\nu X \,.\, \phi \ \vee \ \langle a\rangle X$ exprimem propriedades do tipo "qualquer coisa acontece *até* se atingir um estado em que outra propriedade se verifica". No primeiro caso, porém, este *até* é *estrito*, no sentido em que o estado em causa acaba por ser alcançado num número finito de passos. No segundo o estado que satisfaz ϕ pode nunca ser atingido.

Dois casos particulares desta propriedade são particularmente relevantes na especificação de sistemas concorrentes. O primeiro consiste em substituir a por τ. Neste caso,

$$\mu X \,.\, \phi \ \vee \ \langle \tau \rangle X$$

indica que, após a eventual realização de alguma actividade interna, o processo que satisfaz este requisito alcança um estado que verifica ϕ. Note-se que este é precisamente o significado da fórmula $\langle\!\langle\, \rangle\!\rangle \phi$ discutido atrás e que em $\mathbb{L}_M$ não podíamos introduzir por abreviatura.

O outro caso com interesse surge quando se toma o conjunto *Act* como parâmetro do operador modal. De facto

$$\mu X \,.\, \phi \ \vee \ \langle - \rangle X$$

significa que existe uma computação em que se alcança um estado que verifica a propriedade ϕ após um número finito de transições. Repare-se que esta é uma propriedade de *animação fraca*. *Animação* porque afirma a ocorrência de qualquer coisa (no caso, da evolução para um estado que verifica ϕ) algures num determinado ponto da vida do processo, mas *fraca* porque essa garantia é dada, apenas, relativamente a uma computação. De facto, o processo poderia evoluir através de outra computação que nunca alcançasse o tal estado que valida ϕ.

$\boxed{\textbf{Exemplo: } X \triangleq \phi \ \wedge \ \langle a\rangle X}$

Consideremos, agora, a seguinte equação

$$X \triangleq \phi \ \wedge \ \langle a\rangle X$$

A condição (PRE) toma a forma

$$\text{(PRE)} \qquad \text{Se} \quad E \in \mathbb{P} \text{ e } E \models \phi \wedge \exists_{E' \in X} \ .\ E \xrightarrow{a} E' \quad \text{então} \quad E \in X$$

Fazendo $X = \emptyset$ tornamos falso o antecedente da implicação, pelo que (PRE) resulta verdadeira. Daqui concluimos que $\emptyset$ é um pré-ponto fixo da função que resulta da interpretação em $\mathcal{P}(\mathbb{P})$ do lado direito da equação modal. Claramente a intersecção do conjunto dos pré-pontos fixos será também ela vazia. Assim, $\mu X \,.\, \phi \,\wedge\, \langle a\rangle X$ é equivalente a *ff*. A condição (POST), porém, re-escreve em

$$\text{(POST)} \qquad \text{Se} \quad E \in X \quad \text{então} \quad E \models \phi \wedge \exists_{E' \in X} \,.\, E \xrightarrow{a} E'$$

o que nos faz concluir que o maior ponto fixo, *i.e.*

$$\nu X \,.\, \phi \,\wedge\, \langle a\rangle X$$

será o conjunto de processos que verificam ϕ e, simultaneamente, exibem uma computação infinita da forma

$$E \xrightarrow{a} E_1 \xrightarrow{a} E_2 \xrightarrow{a} E_3 \xrightarrow{a} E_4 \xrightarrow{a} \ldots$$

Dito de outro modo, um processo que verifique esta fórmula exibe a capacidade de realização perpétua da acção a ao longo da qual a propriedade ϕ permanece válida. Este é claramente um requisito de *segurança*.

Poderíamos ser um pouco mais exigentes e requerer ainda que a propriedade ϕ permanecesse válida ao longo de uma computação por a infinita ou *finita*, *i.e.* conducente a um estado terminal do processo. Esta variante toma a forma

$$\nu X \,.\, \phi \,\wedge\, (\langle a\rangle X \vee [a]f\!f)$$

No caso geral, para um conjunto $K \subseteq Act$, exigir que ϕ se verifique durante uma realização *maximal* de acções em K (*i.e.* uma computação infinita ou finita desde que terminal), constitui uma propriedade de *segurança fraca* e exprime-se, como esperavamos, por

$$\nu X \,.\, \phi \,\wedge\, (\langle K\rangle X \vee [K]f\!f)$$

ou, ainda, para $K = Act$, por

$$\nu X \,.\, \phi \,\wedge\, (\langle -\rangle X \vee [-]f\!f)$$

Exemplo: $X \triangleq [-]X$

Vamos terminar com um exemplo um pouco surpreendente. Consideremos a equação

$$X \triangleq [-]X$$

O maior ponto fixo coincide com o próprio $\mathbb{P}$. De facto,

$$\begin{array}{rlll} \text{(POST)} & \text{Se} \quad E \in X & \text{então} & E \in \|[-]\|(X) \\ \equiv & \text{Se} \quad E \in X & \text{então} & (\text{se} \;\; E \xrightarrow{x} E' \text{ e } x \in Act \;\; \text{então} \;\; E' \in X) \end{array}$$

Dito de outro modo, o resultado de uma transição qualquer em qualquer processo que pertença a um post-ponto fixo X, ainda pertence a X. É imediato que o maior subconjunto de $\mathbb{P}$ que verifica esta condição é ele próprio.

Mais curiosa é a análise dos pré-pontos fixos. Temos,

$$\text{(PRE)} \qquad \text{Se} \quad E \in \mathbb{P} \text{ e (se} \;\; E \xrightarrow{x} E' \text{ e } x \in Act \;\; \text{então} \;\; E' \in X) \quad \text{então} \quad E \in X$$

Claramente, o processo $\mathbf{0}$ pertence a qualquer conjunto X que verifique (PRE). Por outro lado, um processo que realize uma qualquer transição e derive em $\mathbf{0}$ pertence igualmente a X, uma vez que $\mathbf{0} \in X$. Mas agora todo o processo que derive por uma qualquer transição noutro que, por sua vez, derive em $\mathbf{0}$, deve ser ainda considerado em X. E assim sucessivamente. O menor X que verifica a condição é então o conjunto de todos os processos finitos. Assim, a propriedade $\mu X \,.\, [-]X$ estabelece, para um processo que a verifique, a impossibilidade de este realizar uma computação infinita. Versões menos fortes deste requisito são obtidas substituindo *Act* por um seu subconjunto K. Nesse caso a propriedade exprime a ausência de computações infinitas com acções em K.

4.6 Taxonomia das Propriedades Temporais

Segurança e Animação

Acabamos de ver como que é possível exprimir propriedades de animação e segurança através, respectivamente, do menor e do maior ponto fixo de determinadas equações. Nos exemplos em causa, tratavam-se de propriedades de animação e segurança *fracas*, na medida em que formulavam uma eventualidade (no primeiro caso) e uma invariância (no segundo) sobre apenas, pelo menos, uma computação. Quer dizer, um processo que verifique qualquer dessas fórmulas pode evoluir realizando computações que, de facto, não validam os requisitos pretendidos.

Recordemos que uma propriedade de animação fraca pode ser expressa pela fórmula

$$\mu X \,.\, \phi \vee \langle - \rangle X$$

Intuitivamente isto significa que existe pelo menos uma computação em que um estado que verifique ϕ pode ser alcançado[2].

Qual será o complementar desta propriedade? Se um processo E não a verifica, então é porque todos os estados, alcançáveis ao longo de qualquer uma das suas computações, verificam a propriedade complementar de ϕ, *i.e.* ϕ^c. Se, por mera cosmética, fizermos $\psi = \phi^c$, tal requisito escreve-se como

$$\nu X \,.\, \psi \wedge [-]X$$

como se pode facilmente verificar. Propriedades deste tipo são ditas de *segurança forte* (repare-se que agora são considerdas *todas* as computações).

Temos, pois, que propriedades de *segurança forte* e de *animação fraca* são complementares. Atrás definimos, estruturalmente, a noção de fórmula complementar. Podemos, agora, alargar essa definição às formulas do tipo $\mu X \,.\, \phi$ e $\nu X \,.\, \phi$. De facto, se uma propriedade é o menor (maior) ponto fixo da equação modal

$$X \triangleq \phi$$

então a propriedade complementar é o maior (menor) ponto fixo da equação

$$X \triangleq \phi^c$$

[2] Como de costume, substituindo *Act* por um qualquer $a \in Act$ ou $K \subseteq Act$, obtemos versões relaxadas da propriedade na medida em que as computações consideradas não são quaisquer mas as geradas por a ou pelas acções em K. Geralmente iremos concentrar a nossa atenção no caso em que $K = Act$, mas tudo o que afirmarmos permanece válido para as versões relaxadas.

Formalmente,

$$(\mu X \,.\, \phi)^{\mathrm{c}} = \nu X \,.\, \phi^{\mathrm{c}}$$
$$(\nu X \,.\, \phi)^{\mathrm{c}} = \mu X \,.\, \phi^{\mathrm{c}}$$

Em exemplo anteriores já deparamos com este tipo de situação. Mostramos, por exemplo, que, enquanto $\nu X \,.\, \langle\tau\rangle X$ exprimia *divergência*, a fórmula complementar garantia *convergência* (*i.e.* comportamento não observável finito). De facto, $(\nu X \,.\, \langle\tau\rangle X)^{\mathrm{c}} = \mu X \,.\, (\langle\tau\rangle X)^{\mathrm{c}} = \mu X \,.\, [\tau]X$.

Vamos aplicar o mesmo raciocínio à fórmula que na acima usamos para exprimir a segurança fraca. Devemos chegar à formulação de uma propriedade que traduza um requisito de *animação* forte. De facto, o complementar de uma garantia de que existe uma computação ao longo da qual ϕ se verifica, é a possibilidade de, em todas as computações, ser possível alcançar um estado que verifique o complementar de ϕ. O complemento de

$$\nu X \,.\, \phi \wedge (\langle-\rangle X \vee [-]\mathit{ff})$$

será

$$\mu X \,.\, \psi \vee ([-]X \wedge \langle-\rangle \mathit{tt})$$

com $\psi = \phi^{\mathrm{c}}$. Esta última fórmula afirma, precisamente, que será alcançado (finitamente) um estado que verifique ψ. Trata-se de uma típica propriedade de animação.

Sumariando, temos que *segurança* e *animação* são propriedades complementares. Não é motivo de surpresa que a primeira recorra ao maior ponto fixo (dada a necessidade de postular uma garantia ao longo de toda uma vida do processo) e a segundo ao menor (porque as "coisas desejáveis" devem ser observáveis em tempo finito).

Acções e Estados

Há, no entanto, uma outra distinção em que é necessário pensar. Consideremos a propriedade de animação forte sobre a Rede de Táxis:

ϕ_0 = *A viatura acaba por regressar à Central*

Se *regressar à Central* fôr modelado pela acção reg, podemos escrever

$$\mu X \,.\, \langle reg\rangle \mathit{tt} \vee ([-]X \wedge \langle-\rangle \mathit{tt})$$

O que aqui se diz, de facto, é que todas as computações do processo viatura alcançam um estado em que é possível realizar a acção reg, facto expresso na sub-fórmula $\langle reg\rangle \mathit{tt}$. Mas isto é *diferente* de afirmar que reg efectivamente ocorre.

O que está em causa é a possibilidade de formularmos as propriedades de animação (forte) sobre os *estados* ou sobre a *ocorrência das acções*. No primeiro caso garantimos que um estado com certas características é alcançado. No segundo, gostaríamos de dizer que uma determinada acção ocorre. No exemplo em mãos diríamos, então,

$$\mu X \,.\, [-reg]X \wedge \langle-\rangle \mathit{tt}$$

A forma geral desta propriedade será, para garantir a realização de uma acção em $K \subseteq Act$,

$$\mu X \,.\, [-K]X \wedge \langle - \rangle tt$$

Por seu lado, a fórmula complementar desta é

$$\nu X \,.\, \langle -K \rangle X \vee [-]ff$$

que, naturalmente, representa uma propriedade de segurança fraca dirigida, não aos estados, como a que acima consideramos, mas à ocorrência das acções.

A relevância desta distinção *estados-acções* pode ser ilustrada pelo seguinte exemplo. Consideremos os processos

$$A_0 \triangleq a.\sum_{i\geq 0} A_i \quad \text{com} \quad A_{i+1} \triangleq b.A_i$$
$$B_0 \triangleq a.\sum_{i\geq 0} B_i + \sum_{i\geq 0} B_i \quad \text{com} \quad B_{i+1} \triangleq b.B_i$$

Não é difícil verificar que o processo $(A_k \mid A_k)$, para um $k > 0$, verifica a propriedade

$$\mu X \,.\, [-a]X \wedge \langle - \rangle tt$$

i.e. garantidamente a acaba por ocorrer. No entanto, falha a propriedade

$$\mu X \,.\, (\langle - \rangle tt \wedge [-a]ff) \ \vee \ (\langle - \rangle tt \wedge [-]X)$$

que afirma ser alcançável um estado em que a seja inevitável. De facto, os dois processos podem evoluir de forma que, em qualquer momento, um deles, pelo menos, ofereça a possibilidade de realizar b. Já o processo $(B_k \mid B_k)$, para um $k > 0$, falha as duas propriedades anteriores, mas verifica

$$\mu X \,.\, \langle a \rangle tt \ \vee \ (\langle - \rangle tt \wedge [-]X)$$

um requisito de animação que afirma ser sempre alcançável um estado em que a é possível (podendo, contudo, nunca ocorrer e, muito menos, surgir como inevitável).

Notemos, por fim, que esta distinção entre propriedades sobre os estados e propriedades sobre as acções não faz sentido quando discutimos propriedades de segurança forte. De facto, dizer que um conjunto K de acções nunca ocorre é equivalente a dizer que todos os estados em qualquer computação verificam $[K]ff$, *i.e.*

$$\nu X \,.\, [K]ff \wedge [-]X$$

Verifica-se imediatamente, por complementação, que a mesma distinção é irrelevante no caso das propriedades de animação fraca.

Propriedades Condicionais

O poder expressivo da lógica modal com pontos fixos é surpreendentemente elevado. Muitas propriedades podem, de facto, ser formuladas combinando as formas básicas estudadas atrás. Por exemplo, suponhamos que queremos exprimir

ϕ_1 = *Após a recolha de um passageiro (icr), o táxi tem de o conduzir ao destino (fcr)*

A segunda parte de ϕ_1 é um requisito de animação forte que exprimimos na forma

$$\mu X \,.\, [-fcr]X \wedge \langle - \rangle tt$$

Porém, esta propriedade só tem de se verificar após o início de uma corrida. Se escrevermos apenas

$$[icr](\mu X \,.\, [-fcr]X \wedge \langle - \rangle tt)$$

estamos a dizer muito pouco — apenas que todas as transições por *icr* a partir do estado inicial conduzem a estados que verificam o referido requisito de animação. O que, em verdade, queremos é especificar que isso mesmo acontece irrelevantemente do estado de evolução do processo Viatura. Ou seja, estamos perante um requisito de segurança forte. Mais exactamente, um requisito de animação *embebido* num requisito de segurança forte. A propriedade ϕ_1 traduz-se, pois, por

$$\nu Y \,.\, [icr](\mu X \,.\, [-fcr]X \wedge \langle - \rangle tt) \;\wedge\; [-]Y$$

Podemos designar este tipo de propriedades por *propriedades condicionais*. Neste caso trata-se de animação condicional, mas podemos igualmente pensar em segurança condicional. Um exemplo é a propriedade que garante que, sempre que uma dada propriedade ϕ se torna verdadeira, outra propriedade ψ não pode deixar de o ser (ou seja, nunca é possível ter-se ψ^{c}). Diríamos, pois,

$$\nu Y \,.\, (\phi^{\mathrm{c}} \vee (\phi \wedge \nu X \,.\, \psi \wedge [-]X)) \wedge [-]Y$$

Outra possibilidade consiste em fixar um requisito de segurança após a eventual ocorrência de uma acção. Por exemplo, e de novo no processo Vaitura da rede de táxis, podemos especificar que

ϕ_2 = *É possível alcançar estados a partir dos quais o regresso à garagem (reg) não pode ocorrer.*

O requisito de segurança é aqui a impossibilidade de regresso que formalizamos como

$$\nu X \,.\, [reg]\mathit{ff} \wedge [-]X$$

Vamos, agora, combina-lo com a propriedade de animação (fraca) que traduz a possibilidade desta situação se verificar. Assim,

$$\mu Y \,.\, (\nu X \,.\, [reg]\mathit{ff} \wedge [-]X) \;\vee\; \langle - \rangle Y$$

Um outro tipo de propridades condicionais que já encontramos são as que se formulam em termos de um *até*. Por exemplo,

$$\mu X \,.\, \psi \vee (\phi \wedge \langle - \rangle tt \wedge [-]X)$$

i.e. em todas as computações ϕ verifica-se até ψ se verificar. Este *até* é estrito, o que significa que ψ acaba por se verificar em todas as computações (note-se que esta situação é distinta da analisada atrás, onde consideramos um *até* estrito mas *fraco*). Esta restrição pode ser removida complementado o ponto fixo, *i.e.*

$$\nu X \,.\, \psi \vee (\phi \wedge \langle - \rangle tt \wedge [-]X)$$

Se retirarmos a parcela $\langle - \rangle tt$, que força o progresso do sistema, obtemos

$$\nu X \, . \, \psi \vee (\phi \wedge [-]X)$$

Ou seja, em todas as computações (incluindo as finitas) ϕ é válido *a menos que* ψ se verifique. Aqui não faz sentido forçar a verificação de ψ.

Propriedades Cíclicas

O tipo mais comum de propriedades que afirmam a recorrência de determinados padrões comportamentais é a classe das propriedades cíclicas. Por exemplo,

ϕ_4 = *Em toda a computação de um determinado processo de gestão de memória, out ocorre em segundo, quarto, sexto, ..., ×2-ésimo lugar.*

traduz-se por

$$\nu X \, . \, [-]([-out]\mathit{ff} \wedge [-]X)$$

Ou, mais realisticamente, poderíamos requerer que a cada *in* se seguisse um *out*, embora outras acções pudessem ocorrer entre estas. Diríamos, então,

$$\mu X \, . \, [out]\mathit{ff} \wedge [in](\mu Y \, . \, [in]\mathit{ff} \wedge [out]X \wedge [-out]Y) \wedge [-in]X$$

Note-se que o uso de menores pontos fixos estabelece a condição adicional de ser finita a quantidade de transições que podem ocorrer entre um *in* e um *out*.

Um pouco mais complexas, mas muito frequentes na especificação de sistemas concorrentes são proprieadades do tipo

ϕ_5 = *Um estado em que in é possível é alcançado um número infinito de vezes*

ou

ϕ_6 = *A acção in pode ocorrer um número infinito de vezes.*

As fórmulas que exprimem estas condições são, respectivamente,

$$\nu X \, . \, \mu Y \, . \, (\langle in \rangle tt \vee \langle - \rangle Y) \;\wedge\; ([-]X \;\wedge\; \langle - \rangle tt)$$

e

$$\nu X \, . \, \mu Y \, . \, [-in]Y \wedge [-]X \wedge \langle - \rangle tt$$

Trocando os operadores de ponto fixo no último exemplo, e substituindo $[-]X \wedge \langle - \rangle tt$ por $[in]X$ obtemos

$$\mu X \, . \, \nu Y \, . \, [-in]Y \wedge [in]X$$

que exprime a possibilidade de *in* ocorrer um número finito de vezes. Note-se que todas estas propriedades são *fortes*, na medida em que realizam uma quantificação implícita sobre todas as computações.

4.7 O μ-calculus Modal

Acabamos de ver que as propriedades temporais dos processos são exprimíveis numa extensão à lógica modal $\mathbb{L}_M$. Tal extensão consiste simplesmente em introduzir dois novos (mas poderosos) operadores que modelam os pontos fixos extremos da interpretação em $\mathcal{P}(\mathbb{P})$ de equações modais. Uma vantagem deste procedimento reside no facto de não termos tido necessidade de introduzir uma nova lógica, nem sequer um novo modelo para a lógica que vinhamos utilizando, surgindo esta como um fragmento bem caracterizado do caso geral.

O "caso geral" é aqui precisamente a lógica $\mathbb{L}_M$ enriquecida com pontos fixos e um conjunto X de variáveis proposicionais. Esta constitui o *μ-calculus modal* [Koz83, Sti95], que aqui vamos designar por $\mathbb{L}_{\mu M}$. Juntemos, pois, os cromos num esforço de sistematização. Assim,

- As fórmulas bem formadas da lógica são-no de acordo com o seguinte BNF:

  $$\phi \ ::= \ X \ | \ \phi_1 \wedge \phi_2 \ | \ \phi_1 \vee \phi_2 \ | \ \langle K \rangle \phi \ | \ [K]\phi \ | \ \mu X \,.\, \phi \ | \ \nu X \,.\, \phi$$

  onde $K \subseteq Act$ e X é um conjunto de variáveis.

  Note-se que as constantes tt e ff podem definir-se por abreviatura como

  $$tt \stackrel{\text{abv}}{=} \nu X \,.\, X \qquad \text{e} \qquad ff \stackrel{\text{abv}}{=} \mu X \,.\, X$$

  Da mesma forma, e como vimos atrás, as modalidades observáveis, antes introduzidas primitivamente em $\mathbb{L}_M$, surgem, agora, como simples abreviaturas de limites. De facto,

  $$\begin{aligned} \langle\!\langle\, \rangle\!\rangle \phi &\stackrel{\text{abv}}{=} \mu X \,.\, \phi \vee \langle \tau \rangle X \\ [\![\ ]\!] \phi &\stackrel{\text{abv}}{=} \nu X \,.\, \phi \wedge [\tau] X \\ [\![\downarrow]\!] \ \phi &\stackrel{\text{abv}}{=} \mu X \,.\, \phi \wedge [\tau] X \end{aligned}$$

  Notemos que a definição de $[\![\ ]\!]\phi$ é complementar da de $\langle\!\langle\, \rangle\!\rangle\phi$. Por outro lado, $[\![\downarrow]\!] \ \phi$ define-se como o *menor* ponto fixo da mesma fórmula usada na definição do operador $[\![\ ]\!]\phi$. Esta escolha garante que nesse caso, ao contrário do que sucede com $[\![\ ]\!]\phi$, é excluida a capacidade de realização infinita de acções não observáveis.

- Tal como a sintaxe, a semântica é apenas uma extensão da noção de modelo que já tínhamos para a lógica modal simples. Seguindo a abordagem atrás esboçada, vamos considerar que a relação de satisfação $\models$ é definida indirectamente em termos dos conjuntos $\|\phi\|$ (*i.e.* os conjuntos de processos que validam a fórmula ϕ).

  Tendo introduzido variáveis na linguagem, é forçoso considerar modelos parametrizados por *valorações*, ou seja, funções que a cada variável atribuem o conjunto de processos em que esta toma o valor tt. Formalmente,

  $$V : X \longrightarrow \mathcal{P}(\mathbb{P})$$

  Obviamente, o cálculo da semântica das fórmulas é relativo à valoração das variáveis livres considerada. Assim, definimos,

  $$\begin{aligned} \|X\|_V &= V(X) \\ \|\phi_1 \wedge \phi_2\|_V &= \|\phi_1\|_V \cap \|\phi_2\|_V \\ \|\phi_1 \vee \phi_2\|_V &= \|\phi_1\|_V \cup \|\phi_2\|_V \\ \|[K]\phi\|_V &= \|[K]\|(\|\phi\|_V) \\ \|\langle K \rangle\phi\|_V &= \|\langle K \rangle\|(\|\phi\|_V) \end{aligned}$$

Até aqui nada de novo: apenas indexamos os conjuntos que interpretam as fórmulas pela valoração V e incluimos uma cláusula própria para dizer que a interpretação de uma variável é o conjunto que a valoração considerada lhe associa. Para o caso das fórmulas cujo conectivo principal é um operador de ponto fixo, teremos, como seria de esperar à luz da discussão anterior,

$$\|\nu X \,.\, \phi\|_V = \bigcup\{S \in \mathbb{P} |\ S \subseteq \|\phi\|_{V[S/X]}\}$$
$$\|\mu X \,.\, \phi\|_V = \bigcap\{S \in \mathbb{P} |\ \|\phi\|_{V[S/X]} \subseteq S\}$$

De facto, toda a fórmula ϕ determina uma função *monótona* entre conjuntos de processos, que a cada $S \subseteq \mathbb{P}$ faz corresponder o conjunto $\|\phi\|_{V[S/X]}$, relativa à valoração V e à variável X. Assim, as interpretações das fórmulas $\nu X \,.\, \phi$ e $\mu X \,.\, \phi$ traduzem, respectivamente, e de acordo com o teorema de Knaster-Tarski, o maior e o menor ponto fixo dessa função.

Notemos, por fim, que o conjunto das fórmulas que não incluem variáveis livres é fechado para a complementação.

A noção de equivalência modal introduzida no início do capítulo permanece válida em $\mathbb{L}_{\mu M}$. Mas o que é muito mais interessante é o facto de os dois resultados sobre a relação entre ela e a bissimulação continuarem a verificar-se. O segundo resultado — *processos com imagem finita, modalmente equivalentes, são bissimilares* — decorre directamente uma vez que a única coisa que fizemos foi estender a lógica. Infelizmente, e ao contrário do que diria alguma (falsa) intuição, a restrição aos processos com imagem finita não pode ser relaxada. O primeiro resultado — *processos bissimilares são modalmente equivalentes* — está exaustivamente demonstrado em [Sti95]. À luz da discussão acima obtemos imediatamente o facto de estes resultados permanecerem válidos para o fragmento da lógica baseado nas modalidades observáveis.

NOTA: Terminamos aqui o estudo das lógicas para processos. Foi nossa opção não dar uma panorâmica das inúmeras abordagens disponíveis, mas estudar uma delas com alguma profundidade. O mesmo fizemos, de resto, na primeira parte da disciplina em que reduzimos o estudo das álgebras de processos ao estudo de CCS.

A lógica estudada — o μ-calculus modal — foi inicialmente proposta em [Koz83], aplicando ao caso modal uma extensão que se tem revelado fecunda em outro tipo de lógicas: a inclusão de operadores de ponto fixo. Trata-se de uma lógica finitária e proposicional que apresenta duas vantagens importantes:

- é *decidível* e
- dotada de um grande poder expressivo (estritamente mais expressiva que lógicas como PDL e CTL* muito populares nas ciências da computação).

Entre os vários tópicos que não puderam ser abordados neste curso, encontra-se um que é de vital importância: o estudo dos métodos de prova para o μ-calculus modal. De facto, o cálculo de pontos fixos pelo teorema de Knaster-Tarski torna-se pouco expedito, e de difícil compreensão, para fórmulas em que surgem diversos conectivos μ e ν. O seu cálculo mais sistemático pode ser feito calculando os limites de cadeias de Kleene, uma vez que as funções que interpretam as fórmulas modais são, não apenas monótonas, mas também, em certas condições, *contínuas*. O processo, exemplificado com detalhe em [Sti95], é instrutivo (na medida em que o próprio cálculo das aproximações aumenta a nossa compreensão intuitiva do significado da fórmula), embora para processos não finitos, os índices das cadeias tenham que se estender para lá do ordinal ω. Em muitos casos, em particular para análise de processos com estado finito, existem procedimentos muito expeditos de verificação de fórmulas do μ-calculus modal que são variantes de algoritmos de *model checking*. O CWB [MS96] implementa um destes algoritmos.

Exercícios

Exercício III.25

Formule cada uma das seguintes propriedades na lógica modal $\mathbb{L}_M$. Repare que algumas são propositadamente vagas e podem, portanto, ser formalizadas de diversos modos.

1. A ocorrência de um a e um b é impossível.
2. A ocorrência de um a seguido de um b é impossível.
3. Apenas a ocorrência de a é possível.
4. Após a ocorrência de a é possível a ocorrência de b ou c.
5. Após a ocorrência de a ou b é possível a ocorrência de c.
6. Se a ocorrer, é possível a ocorrência de b ou c, mas não de ambos.
7. a não pode ocorrer antes de b.
8. Existe apenas uma transição etiquetada por a.

Exercício III.26

Considere os processos seguintes e indique, para cada um deles, quais das propiedades acima são verificadas:

1. $E_1 \triangleq a.b.\mathbf{0}$
2. $E_2 \triangleq a.c.\mathbf{0}$
3. $E \triangleq E_1 + E_2$
4. $F \triangleq a.(b.\mathbf{0} + c.\mathbf{0})$
5. $G \triangleq E + F$

Exercício III.27

Considere a seguinte especificação de um dispositivo de corte de madeira.

$$\begin{aligned} Start &\triangleq fw.Go + stop.\mathbf{0} \\ Go &\triangleq fw.bk.bk.Start + right.left.bk.Start \end{aligned}$$

Escreva em $\mathbb{L}_M$ as seguintes propriedades:

1. Após fw outro fw é imediatamente possível.
2. Após um fw seguido de um $right$, $left$ é possível e bk não.
3. A acção fw é, inicialmente, a única possível.
4. A terceira acção de $Start$ é distinta de fw.

Exercício III.28

Seja $P \triangleq (A \mid B)\backslash_{\{a\}} + a.\mathbf{0}$, onde $A \triangleq \underline{fix}\ (X = a.X)$ e $B \triangleq \underline{fix}\ (X = \overline{a}.X)$. Verifique ou refute as seguintes afirmações:

1. $P \models \langle a \rangle tt \vee \langle \overline{a} \rangle tt$

2. $P \models [\![a]\!][-]ff$
3. $P \models [\![\downarrow a]\!][-]ff$
4. $P \models [\![a \downarrow]\!][-]ff$
5. $P \models [\![\downarrow]\!]tt$

Exercício III.29

A descrição que se segue modela um sistema de comunicação entre um processo S, que aceita pedidos para enviar mensagens, e um processo R, que as recebe. O meio de comunicação é simulado pelo processo M e está longe de ter um comportamento fiável.

$$S \triangleq send.\overline{s}.S$$
$$R \triangleq r.\overline{receive}.S$$
$$M \triangleq s.\overline{r}.M + s.\overline{r}.\overline{r}.M + s.M$$

$$P \triangleq (S \mid M \mid R)\backslash_{\{e,r\}}$$

1. Explique porque razão o meio de comunicação não é fiável, identificando os problemas que podem surgir quando é solicitada a transmissão de uma mensagem.
2. Mostre que $(\overline{s}.S \mid s.M \mid R)\backslash_{\{e,r\}} \xrightarrow{\tau} P$ é uma transição possível.
3. Verifique que $P \models [send]\langle\!\langle\overline{receive}\rangle\!\rangle tt$.
4. Para lidar com um meio de comunicação tão pouco fiável seria necessário especificar um protocolo de comunicação muito mais robusto. Tal protocolo deveria, em particular, garantir que *toda a mensagem enviada fosse recebida*. Escreva uma fórmula em $\mathbb{L}_M$ que registe formalmente essa propriedade.

Exercício III.30

Seja E um processo. Uma fórmula ϕ diz-se a *fórmula característica* de E sse

$$\forall_{F \in \mathbb{P}} \,.\, F \models \phi \;\; sse \;\; F \sim E$$

Repare que, por definição, um processo verifica a *fórmula característica* de E sse fôr estritamente equivalente a E. Determine a *fórmula característica* do processo $x.\mathbf{0}$.

Exercício III.31

Considere os processos $E \triangleq a.(b.\mathbf{0} + c.\mathbf{0})$ e $F \triangleq a.b.\mathbf{0} + a.c.\mathbf{0}$. Escreva uma fórmula ϕ em $\mathbb{L}_M$ que seja válida em E mas não em F.

Exercício III.32

Considere os processos seguintes e escreva uma fórmula em $\mathbb{L}_M$ que seja válida para o proceso R e falsa para S.

$$E \triangleq b.c.\mathbf{0} + b.d.\mathbf{0} \tag{4.1}$$
$$F \triangleq E + b.(c.\mathbf{0} + d.\mathbf{0}) \tag{4.2}$$
$$R \triangleq a.E + a.F \tag{4.3}$$
$$S \triangleq a.F \tag{4.4}$$

Exercício III.33

Defina em $\mathbb{L}_M$, por abreviatura, um operador (K), com $K \subseteq Act$, de forma que $E \models (K)\phi$ sse as acções em K forem as acções iniciais de E, todas elas conduzindo a estados que validam ϕ.

Exercício III.34

Como sabe, a composição paralela não é, no caso geral, idempotente.

1. Fazendo $E \triangleq a.b.E$, formule uma propriedade em $\mathbb{L}_M$ que distinga E de $E \mid E$.
2. Em certos casos, porém, a idempotência verifica-se. Construa uma bissimulação que demonstre a equivalência $E \sim E \mid E$ quando E é da forma $\underline{fix}\ (X = \sum_{x \in K} x.X)$, para um qualquer $K \subseteq L$. Será a conclusão anterior válida para todo o $K \subseteq \overline{Act}$?

Exercício III.35

Seja P um processo, a uma acção e ϕ uma fórmula em $\mathbb{L}_M$. Assumindo que $P \models \langle\langle a \rangle\rangle\, \phi$, indique justificando quais das asserções seguintes são verdadeiras.

1. $a.P \models \langle\langle a \rangle\rangle\, \phi$
2. $\tau.P \models \langle\langle a \rangle\rangle\, \phi$
3. $P[b/a] \models \langle\langle b \rangle\rangle\, \phi$, para qualquer acção b diferente de τ
4. $P \mid Q \models \langle\langle a \rangle\rangle\, \phi$, para qualquer processo Q
5. $P + Q \models \langle\langle a \rangle\rangle\, \phi$, para qualquer processo Q

Exercício III.36

Determine

1. $\|[a][b]\langle c, d\rangle tt\|$
2. $\|\langle a\rangle\langle -\rangle tt\|$
3. $\|[a]\langle -\rangle tt \wedge [b][-]f\!f\|$
4. $\|[a]\langle -\rangle tt \vee [b][-]f\!f\|$

Exercício III.37

O reconhecimento de linguagens regulares baseia-se, como sabe, na noção de *autómato*. Basicamente, este consiste num conjunto Σ de símbolos (o alfabeto), um conjunto finito S de estados, um estado onde se inicia o reconhecimento, uma relação de transição $\delta \subseteq S \times \Sigma \times S$ e um conjunto F de estados finais correspondentes às sequências de símbolos aceites pelo autómato. Por exemplo, o autómato especificado por

$$\Sigma = \{a, b\} \quad S = \{1, 2\} \quad s_0 = 1 \quad F = \{2\}$$
$$\delta = \{(1, a, 1), (1, b, 2), (2, a, 2), (2, b, 1)\}$$

reconhece a linguagem formada por sequências de a e b, com um número ímpar de bs.

1. Codifique o autómato descrito acima numa expressão da linguagem $\mathbb{P}$ de processos que estudou.
2. Mostre ou refute (fornecendo contra-exemplos adequados) que
3. Dois autómatos distintos que reconhecem a mesma linguagem regular são observacionalmente equivalentes.
4. Dois autómatos que verificam um mesmo conjunto de propriedades expressas em $\mathbb{L}_M$, são estritamente equivalentes.
5. Dois autómatos que verificam um mesmo conjunto de propriedades expressas em $\mathbb{L}_{\mu M}$, são estritamente equivalentes.

Exercício III.38

Um sistema de alarme residencial é suposto fazer soar um alarme (acção modelada por alm) logo que detecta a presença de um intruso (situação modelada por int).

1. Será que a fórmula $[int](\langle alm\rangle tt \ \wedge \ [-alm]ff)$ em $\mathbb{L}_M$ representa adequadamente essa propriedade comportamental?
2. Caso pense que não, represente-a correctamente (porventura em $\mathbb{L}_{\mu M}$)

Exercício III.39

Suponha que num processo que especifica o comportamento de uma máquina de azar a acção $ganha(x)$ modela o facto do jogador ganhar uma quantia de x moedas. Alguém sugeriu que o processo deveria satisfazer uma das seguintes propriedades:

$$\begin{aligned}\phi_1 &= \nu X\,.\,(\mu Y\,.\,(\langle ganha(1000)\rangle tt \vee \langle -\rangle Y) \wedge [-]X)\\ \phi_2 &= \nu X\,.\,\mu Y\,.\,(\langle ganha(1000)\rangle X \vee \langle -\rangle Y\end{aligned}$$

Alguém, porém, argumentou que ϕ_1 e ϕ_2 eram equivalentes.

1. Explique o significado destas propriedades e discuta se serão ou não equivalentes.
2. Recorde a classificação das propriedades modais e temporais. Em que classe incluiría ϕ_1? E ϕ_2? Justifique.

Exercício III.40

Formule em $\mathbb{L}_{\mu M}$ as seguintes propriedades sobre o comportamento de uma rede de táxis (*cf.*, exercício I.20) e classifique-as:

1. Toda a viatura acaba por regressar à Central.
2. Toda a viatura em serviço acaba por regressar à Central.
3. Para toda a viatura em serviço existe uma computação que a conduz a um estado em que pode regressar à Central.
4. Após a recolha de um passageiro, a corrida tem de terminar.
5. Uma vez alocada a um serviço, a viatura não pode regressar sem que este seja anulado ou completado com sucesso.
6. Qualquer viatura pode detectar um cliente na rua.

Exercício III.41

Formule em $\mathbb{L}_{\mu M}$ a propriedade seguinte sobre o comportamento de uma máquina de venda automática de bebidas: *O depósito de uma ou duas fichas conduz à aquisição de um café ou um chá.*

Exercício III.42

Uma propriedade importante em sistemas que controlam linhas de montagem indutriais é a garantia de que

ϕ = *sempre que uma situação de erro grave ocorre, o sistema pára.*

Note, porém, que, regra geral, não pára instantaneamente: por exemplo, pode ser necessário que, antes de parar, o sistema desligue certos circuitos, active indicadores luminosos num painel, etc.

1. Supondo que a acção *erro* modela a ocorrência de um erro grave, codifique a propriedade ϕ em $\mathbb{L}_{\mu M}$.
2. Recorde a classificação das propriedades modais e temporais. Em que classe incluiría ϕ? Justifique.

Exercício III.43

Considere a definição por abreviatura do seguinte operador sobre processos:

$$\delta_K P \stackrel{\text{abv}}{=} \underline{fix}\ (X = \sum_{a \in K} a.X + P)$$

onde $K \subseteq Act$ e $P \in \mathbb{P}$.

1. Explique informalmente o significado deste operador e imagine um exemplo de utilização.
2. Mostre que, se $K = K'$, $\delta_K \delta_{K'} P \sim \delta_K P$.
3. Como poderá simplificar $\delta_K \delta_{K'} P$ no caso geral em que $K \neq K'$?
4. Utilizando o teorema da expansão, indique porque razão, para qualquer processo P, o processo $\delta_{\{\overline{a},b\}} P \mid \delta_{\{\overline{c},a\}} \mathbf{0}$ é divergente.
5. Formule em $\mathbb{L}_{\mu M}$ uma propriedade que exprima a possibilidade de divergência. Indique se trata de uma propriedade modal ou temporal.
6. Suponha que $P \models \phi$. Discuta se as propriedades $\mu X \,.\, [K]X$ e $\nu X \,.\, \langle K \rangle X \vee \phi$ são verificadas pelo processo $\delta_K P$.
7. Repita a alínea anterior considerando, agora, o processo $\delta_{K'} P$, com $K \subseteq K'$.

Exercício III.44

Quando um número limitado de recursos é partilhado por um número elevado de processos existe a possibilidade de alguns desses processos clientes terem de aguardar permissão para utilizar um recurso até que este seja libertado pelo processo que o está correntemente a usar. A alocação de recursos aos clientes é um problema típico, por exemplo, na concepção de sistemas operativos, onde é conhecido como o problema de *scheduling*.

Para que a alocação seja feita com sucesso é necessário saber quais os processos que aguardam permissão de acesso. Assim, o acesso não pode ser modelado por uma única acção atómica, mas antes por um par de acções: *request* (para requerer o acesso) e *permission* (para indicar que este foi concedido). Quando o recurso é libertado o processo realiza a acção *release* para sinalizar que terminou a tarefa e que o recurso está de novo disponível.

Um algoritmo de alocação muito popular é similar ao usado em certas lojas em que cada utilizador tira um *ticket* com um número, emitido, por ordem estritamente crescente, por um dispositivo próprio. Os recursos partilhados são, neste exemplo, os funcionários da loja e, usualmente, existe um número fixo de funcionários que podem atender clientes simultaneamente. Sempre que um deles está livre chama o cliente com o número mais baixo e que ainda aguarda ser servido.

1. Suponha que se pretende modelar uma situação em que existem R recursos (similares) disponíveis, o que significa que R processos podem ser "atendidos" ao mesmo tempo. Complete a especificação do processo:

$$\begin{aligned} Sch &\triangleq Sch(0,0,0) \\ Sch(p,t,r) &\triangleq \text{if} \quad \ldots \quad \text{then } \overline{permission_t} \,.\quad \ldots \\ &\qquad \text{else } \overline{request(p)} \,.\quad \ldots + \quad release(x) \,.\quad \ldots \end{aligned}$$

   que realiza a alocação de processos a recursos de acordo com o protocolo acima descrito. O processo tem 3 parâmetros:

   - p é o número a atribuir ao próximo processo que solicite acesso (coincidente com o número de processos que já realizaram *request*);
   - t é o número do próximo processo a ser atendido (coincidente com o número de processos que já realizaram *permission*);
   - r é o número de processos que já foram servidos e libertaram o recurso utilizado realizando *release*.

2. Os 3 contadores — p, t e r — podem, eventualmente, acabar por atingir o mesmo valor $n > 0$. Diga quando é que tal situação pode ocorrer e modifique a definição de Sch de modo a que, nessas circunstâncias, os contadores sejam re-inicializados com o valor 0.

3. Um dos objectivos deste protocolo é garantir que nunca existe, simultaneamente, um processo à espera de permissão de acesso e um recurso livre. Especifique essa propriedade em $\mathbb{L}_{\mu M}$.

4. Outra propriedade que é desejável garantir é a ausência de *esperas infinitas*, *i.e.*, todo o processo cliente que pede permissão acaba por conseguir aceder ao recurso. Especifique essa propriedade em $\mathbb{L}_{\mu M}$ e diga como poderia demonstrar que Sch a verifica.

Referências

[AS85] B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, (21):181–185, 1985.

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

[BK85] J. Bergstra and J. Klop. Algebra for communicating processes with abstraction. *Theoretical Computer Science*, (37):77–121, 1985.

[BRV95] P. Blackburn, M. de Rijke, and I. Venema. The algebra of modal logic. CWI Tech. Rep., CWI, Amsterdam, 1995.

[Dij76] E. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[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.

[HM85] M. C. Hennessy and A. J. R. G. Milner. Algebraic laws for non-determinism and concurrency. *Journal of ACM*, 32(1):137–161, 1985.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Comm. of ACM*, (12):576–580, 1969.

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

[Jon86] Cliff B. Jones. *Systematic Software Development Using* VDM. Series in Computer Science. Prentice-Hall International, 1986.

[JT51] B. Jónsson and A. Tarski. Boolean algebras with operators. *American Journal of Mathematics*, 78:891–939, 1951.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, (27):333–353, 1983.

[Kri63] S. Kripke. Semantical analysis of modal logic I: normal propositional calculi. *Zeitschrift fur mathematische Logik und Grundlagen der Mathematik*, (9):67–96, 1963.

[Lam77] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. Software Engineering*, (3):125–143, 1977.

[Lam83] L. Lamport. What good is temporal logic? In R. E. A. Mason, editor, *Proc. IFIP 9th World Congress*, pages 657–668. North Holland, 1983.

[LS91] K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, (94):1–28, 1991.

[Mil83] A. J. R. G. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, (25):794–818, 1983.

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

[MP90] Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *Proc. 9th ACM Symp. on Principles of Distributed Computing*, pages 377–408, 1990.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer Verlag, 1992.

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

[MS96] F. Moller and P. Stevens. The edinburgh concurrency workbench (version 7). User's manual, LFCS, Edinburgh University, 1996.

[Oli90] J. N. Oliveira. A reification calculus for model-oriented software specification. *Formal Aspects of Computing*, 2(1):1–23, 1990.

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

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th Annual Symp. on Foundations of Computer Science*, pages 46–57, 1977.

[Pra92] V. Pratt. Origins of the calculus of binary relations. In *Proc. 7th Annual IEEE Symp. on Logic in Computer Science*, pages 248–254. Santa Cruz, CA, 1992.

[PT97] B. Pierce and D. N. Turner. Pict: A programming language based in the π-calculus. In MIT Press, editor, *Milner Festschrift (to appear).* Elsevier Science Publishers B. V. (North-Holland), 1997.

[Rei88] W. Reisig. Temporal logic and causality in concurrent systems. In F. H. Vogt, editor, *Concurrency 88.* LNCS (335), 1988.

[Rik93] M. de Rikje. *Extended Modal Logic.* PhD thesis, University of Amsterdam, 1993.

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

[Sti92] C. Stirling. Modal and temporal logics. In Maibaum Abramsky, Gabbay, editor, *Handbook of Logic in Computer Science (vol. 2)*, pages 478–551. Oxford Science Publications, 1992.

[Sti95] C. Stirling. Modal and temporal logics for processes. *Springer Lect. Notes Comp. Sci. (715)*, pages 149–237, 1995.

[Sto36] M. Stone. The Theory of Representation for Boolean Algebras. *Trans. Amer. Math. Soc.*, (40):37–111, 1936.

[Tar55] A. Tarski. A lattice–theoretic fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[WN95] G. Winskel and M. Nielsen. Models for Concurrency. In S. Abramsky, D. M. Gabbay, and T. S. E. Gabbay, editors, *Handbook of Logic in Computer Science (vol. 4))*, pages 1–148, 1995.